



Guide du développeur

AWS SDK pour .NET (V3)



AWS SDK pour .NET (V3): Guide du développeur

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Les marques commerciales et la présentation commerciale d'Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d'une manière susceptible d'entraîner une confusion chez les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

.....	xiii
Qu'est-ce que le AWS SDK pour .NET	1
À propos de cette version	1
Maintenance et prise en charge des versions majeures du SDK	2
Cas d'utilisation courants	2
Sujets supplémentaires dans cette section	2
AWS Outils connexes	3
Outils pour Windows PowerShell et outils pour PowerShell Core	3
Toolkit for VS Code	3
Toolkit for Visual Studio	3
Boîte à outils pour Azure DevOps	4
Plateformes prises en charge	4
.NET Core	4
.NET Standard 2.0	5
.NET Framework 4.5	5
.NET Framework 3.5	5
Bibliothèque de classes portable et Xamarin	6
Support Unity	6
En savoir plus	6
SDKs et référence sur les outils	6
Historique des révisions	6
Nouveautés	7
Ressources supplémentaires	9
Mise en route	12
Installez et configurez votre chaîne d'outils	12
Développement multiplateforme	13
Windows avec Visual Studio et .NET Core	13
Étape suivante	14
Configurer l'authentification du SDK	14
Connectez-vous en utilisant les informations d'identification de console	14
Activation et configuration d'IAM Identity Center	16
Configurez le SDK pour utiliser IAM Identity Center.	16
Démarrer une session sur le portail AWS d'accès	17
Informations supplémentaires	18

Faites une visite rapide	18
Application multiplateforme simple	19
Application simple basée sur Windows	25
Étapes suivantes	31
Lancement d'un nouveau projet	31
Configuration de la AWS région	32
Création d'un client de service avec une région particulière	33
Spécifiez une région pour tous les clients du service	34
Résolution régionale	35
Informations spéciales sur la région de Chine (Pékin)	36
Informations spéciales sur les nouveaux AWS services	36
Installez AWSSDK des packages avec NuGet	36
Utilisation NuGet à partir de l'invite de commande ou du terminal	37
Utilisation NuGet depuis l'explorateur de solutions Visual Studio	37
Utilisation NuGet depuis la console Package Manager	38
Installez AWSSDK des assemblages sans NuGet	39
Résolution des informations d'identification et des profils	40
Résolution du profil	41
Utilisation des informations d'identification d'un compte utilisateur fédéré	42
Spécifier des rôles ou des informations d'identification temporaires	43
Utilisation des informations d'identification du proxy	43
Utilisateurs et rôles	44
Utilisateurs et ensembles d'autorisations	44
Rôles de service	45
Configuration avancée	46
AWSSDK.Prolongations. NETCore.Configuration et IConfiguration	46
Configuration d'autres paramètres de l'application	51
Référence des fichiers de configuration pour AWS SDK pour .NET	59
Utilisation d'informations d'identification existantes	71
Avertissements et conseils importants concernant les informations d'identification	72
Utilisation du fichier AWS d'informations d'identification partagé	73
Utilisation du SDK Store (Windows uniquement)	77
Fonctionnalités du SDK	82
Asynchrone APIs	82
Rétentatives et délais d'attente	84
Nouvelles tentatives	84

Délais	86
Paginateurs	89
Où puis-je trouver des paginateurs ?	89
Que m'apportent les paginateurs ?	89
Pagination synchrone ou asynchrone	90
exemple	90
Considérations supplémentaires pour les paginateurs	94
Observabilité	95
Ressources supplémentaires	95
Configurez un TelemetryProvider	95
Métriques	97
Fournisseurs de services de télémétrie	99
Outils supplémentaires	101
AWS Outil de déploiement	101
AWS Framework de traitement des messages pour .NET	101
Intégrations avec .NET Aspire	101
Authentification avancée	102
Authentification unique	102
Prérequis	103
Configuration d'un profil SSO	103
Génération et utilisation de jetons SSO	105
Ressources supplémentaires	110
Didacticiels	110
Tutoriel : application .NET uniquement	111
Tutoriel : AWS CLI et application .NET	120
Déployer vers AWS	129
Déploiement à partir de la CLI .NET	129
Déployez à partir des boîtes à outils de l'IDE	129
Cas d'utilisation	130
Applications ASP.NET Core	130
Applications de console .NET	131
Applications Blazor WebAssembly	132
AWS Lambda projets	133
Conditions préalables	133
Commandes Lambda disponibles	134
Étapes de déploiement	134

Migrez votre projet	136
Migration vers la version 3	136
À propos des AWS SDK pour .NET versions	136
Refonte de l'architecture du kit SDK	136
Évolutions	136
Migration vers la version 3.5	138
Ce qui a changé pour la version 3.5	139
Migration de code synchrone	140
Migration vers la version 3.7	141
Migration à partir de .NET Standard 1.3	141
Travaillez avec les AWS services	143
Exemples de code avec conseils	143
CloudFormation	144
Amazon Cognito	148
DynamoDB	156
Amazon EC2	186
IAM	249
Amazon S3	269
Amazon SNS	279
Amazon SQS	283
AWS Lambda	317
APIs	317
Prérequis	317
Informations supplémentaires	317
Rubriques	317
Annotations Lambda	318
Bibliothèques et frameworks de haut niveau	319
Cadre de traitement des messages	320
Intégration AWS à .NET Aspire	340
Autres services et configuration	342
Exemples de code	343
ACM	345
Actions	345
API Gateway	349
Scénarios	350
AWS contributions communautaires	351

Aurora	351
Scénarios	350
Auto Scaling	352
Actions	345
Scénarios	350
Exécution d'Amazon Bedrock	404
Scénarios	350
Amazon Nova	420
Amazon Nova Canvas	440
Anthropic Claude	442
Cohere Command	444
Meta Llama	448
Mistral AI	452
CloudWatch Journaux	456
Actions	345
Fournisseur d'identité Amazon Cognito	470
Actions	345
Scénarios	350
Amazon Comprehend	495
Actions	345
Scénarios	350
Amazon DocumentDB	507
Exemples sans serveur	507
DynamoDB	510
Actions	345
Scénarios	350
Exemples sans serveur	507
AWS contributions communautaires	351
Amazon EC2	577
Principes de base	578
Actions	345
Scénarios	350
Amazon ECS	701
Actions	345
Scénarios	350
Elastic Load Balancing – version 2	713

Actions	345
Scénarios	350
EventBridge	770
Mise en route	771
Principes de base	578
Actions	345
EventBridge Planificateur	811
Mise en route	771
Actions	345
Scénarios	350
Amazon Glacier	841
Mise en route	771
Actions	345
AWS Glue	851
Mise en route	771
Principes de base	578
Actions	345
IAM	883
Mise en route	771
Principes de base	578
Actions	345
Scénarios	350
Amazon Keyspaces	981
Mise en route	771
Principes de base	578
Actions	345
Kinesis	1009
Actions	345
Exemples sans serveur	507
AWS KMS	1027
Actions	345
Lambda	1039
Mise en route	771
Principes de base	578
Actions	345
Scénarios	350

Exemples sans serveur	507
AWS contributions communautaires	351
MediaConvert	1091
Mise en route	771
Actions	345
Amazon MSK	1101
Exemples sans serveur	507
Organizations	1103
Actions	345
Partner Central	1122
Actions	345
Amazon Pinpoint	1128
Actions	345
Amazon Polly	1135
Actions	345
Scénarios	350
Amazon RDS	1148
Mise en route	771
Principes de base	578
Actions	345
Scénarios	350
Exemples sans serveur	507
Amazon RDS Data Service	1187
Scénarios	350
Amazon Rekognition	1188
Actions	345
Scénarios	350
Enregistrement de domaine Route 53	1220
Mise en route	771
Principes de base	578
Actions	345
Amazon S3	1247
Actions	345
Scénarios	350
Exemples sans serveur	507
SageMaker IA	1389

Mise en route	771
Actions	345
Scénarios	350
Secrets Manager	1424
Actions	345
Amazon SES	1427
Actions	345
Scénarios	350
API Amazon SES v2	1442
Actions	345
Scénarios	350
Amazon SNS	1481
Mise en route	771
Actions	345
Scénarios	350
Exemples sans serveur	507
Amazon SQS	1527
Mise en route	771
Actions	345
Scénarios	350
Exemples sans serveur	507
Step Functions	1571
Mise en route	771
Principes de base	578
Actions	345
AWS STS	1599
Actions	345
Support	1601
Mise en route	771
Principes de base	578
Actions	345
Amazon Textract	1629
Scénarios	350
Amazon Transcribe	1631
Actions	345
Amazon Translate	1643

Actions	345
Scénarios	350
Sécurité	1656
Protection des données	1657
Gestion de l'identité et des accès	1658
Public ciblé	1658
Authentification par des identités	1659
Gestion de l'accès à l'aide de politiques	1660
Comment Services AWS travailler avec IAM	1662
Résolution des problèmes AWS d'identité et d'accès	1663
Validation de la conformité	1665
Résilience	1665
Sécurité de l'infrastructure	1666
Application d'une version minimale de TLS	1666
.NET Core	1667
.NET Framework	1668
Outils AWS pour PowerShell	1669
Xamarin	1670
Unity	1670
Navigateur (pour Blazor WebAssembly)	1670
Migration du client de chiffrement S3 (V1 vers V2)	1671
Présentation de la migration	1671
Mettez à jour les clients existants vers des clients de transition vers la version 1 pour lire les nouveaux formats	1672
Migrer les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats	1673
Mettre à jour les clients V2 pour ne plus lire les formats V1	1676
Migration du client de chiffrement S3 (V2 vers V4)	1677
Comprendre les concepts de la V4	1677
Mettre à jour les clients existants pour lire les formats V4	1679
Migrer vers des clients V4	1680
Exemples de configuration supplémentaires	1684
Considérations spéciales	1687
Obtention d' AWSSDK assemblages	1687
Téléchargez et extrayez des fichiers ZIP	1687
Accès aux informations d'identification et aux profils dans une application	1688
Exemples de cours CredentialProfileStoreChain	1689

Exemples pour les classes SharedCredentialsFile et AWSCredentials Factory	1691
Support Unity	1691
Assistance Xamarin	1693
Référence d'API	1694
À propos des versions de référence de l'API	1694
Historique du document	1697

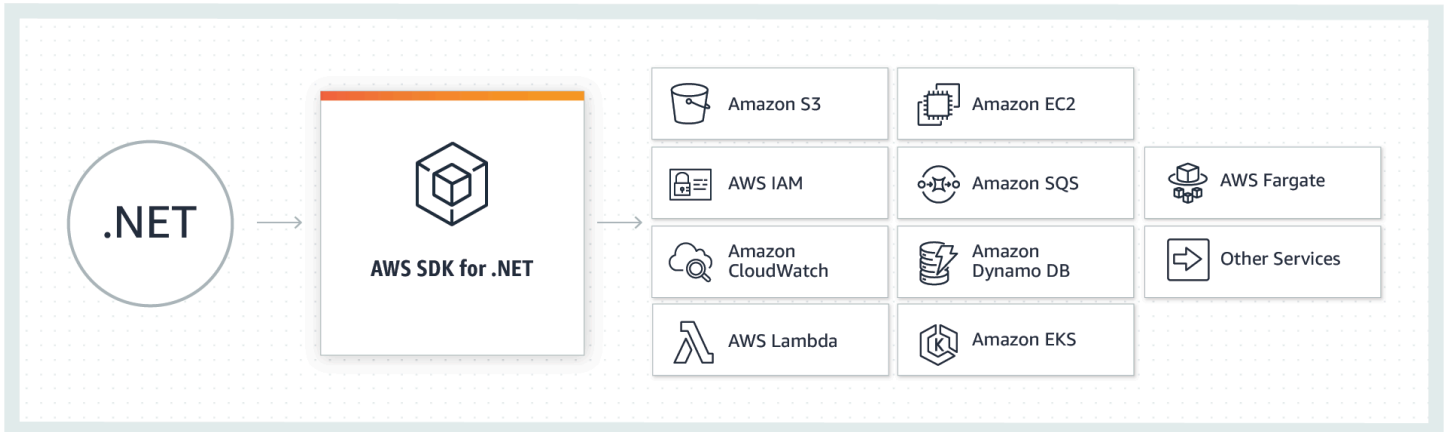
La AWS SDK pour .NET V3 est passée en mode maintenance.

Nous vous recommandons de migrer vers la version [AWS SDK pour .NET V4](#). Pour plus de détails et d'informations sur la façon de migrer, veuillez consulter notre [annonce relative au mode de maintenance](#).

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.

Qu'est-ce que le AWS SDK pour .NET

AWS SDK pour .NET Cela facilite la création d'applications .NET qui exploitent des AWS services rentables, évolutifs et fiables tels qu'Amazon Simple Storage Service (Amazon S3) et Amazon Elastic Compute Cloud (Amazon EC2). Le SDK simplifie l'utilisation des AWS services en fournissant un ensemble de bibliothèques cohérentes et familières aux développeurs .NET.



(OK, j'ai compris ! Je suis prêt à m'[installer et à faire une visite rapide](#).)

À propos de cette version

Documentation actuelle et passée

Cette documentation concerne la version 3 (V3) du AWS SDK pour .NET. La V3 n'est pas la dernière version du SDK. Pour obtenir la dernière version du SDK, consultez le [guide du développeur AWS SDK pour .NET \(V4\)](#).

Cette documentation est principalement centrée sur .NET Core et ASP.NET Core, mais contient également des informations sur .NET Framework et ASP.NET 4. x. Outre Windows et Visual Studio, il accorde une attention égale au développement multiplateforme.

Pour plus d'informations sur la migration à partir de versions antérieures du SDK, consultez [Migrez votre projet](#)

Pour trouver le contenu obsolète des versions antérieures du SDK, consultez les éléments suivants :

- [AWS SDK pour .NET \(version 2, obsolète\) Guide du développeur](#)

- [Références d'API obsolètes pour AWS SDK pour .NET](#)

Maintenance et prise en charge des versions majeures du SDK

Pour plus d'informations sur la maintenance et le support des versions majeures du SDK et de leurs dépendances sous-jacentes, consultez les informations suivantes dans le [guide de référence AWS SDKs and Tools](#) :

- [AWS SDKs et politique de maintenance des outils](#)
- [AWS SDKs et matrice de support des versions d'outils](#)

Cas d'utilisation courants

AWS SDK pour .NET Cela vous aide à réaliser plusieurs cas d'utilisation convaincants, notamment les suivants :

- Gérez les utilisateurs et les rôles avec [Gestion des identités et des accès AWS \(IAM\)](#).
- Accédez à [Amazon Simple Storage Service \(Amazon S3\)](#) pour créer des buckets et stocker des objets.
- Gérez les abonnements HTTP [Amazon Simple Notification Service \(Amazon SNS\)](#) aux rubriques.
- Utilisez l'[utilitaire de transfert S3](#) pour transférer des fichiers vers Amazon S3 depuis vos applications Xamarin.
- Utilisez [Amazon Simple Queue Service \(Amazon SQS\)](#) pour traiter les messages et les flux de travail entre les composants d'un système.
- Effectuez des transferts Amazon S3 efficaces en envoyant des instructions SQL à [Amazon S3 Select](#).
- Créez et lancez EC2 des instances [Amazon](#), configurez et demandez des [instances Amazon EC2 Spot](#).

Sujets supplémentaires dans cette section

- [AWS outils liés à AWS SDK pour .NET](#)

- [Plateformes soutenues par le AWS SDK pour .NET](#)
- [AWS SDKs Guide de référence et d'outils](#)
- [Historique des révisions](#)
- [Quoi de neuf dans le AWS SDK pour .NET](#)
- [Ressources supplémentaires](#)

AWS outils liés à AWS SDK pour .NET

Outils pour Windows PowerShell et outils pour PowerShell Core

Les AWS Tools for Windows PowerShell et AWS Tools for PowerShell Core sont PowerShell des modules basés sur les fonctionnalités exposées par le AWS SDK pour .NET. Les AWS PowerShell outils vous permettent de créer des scripts pour des opérations sur vos AWS ressources à partir de l' PowerShell invite. Bien que les applets de commande soient implémentés à l'aide des clients de service et des méthodes du SDK, les applets de commande fournissent une PowerShell expérience idiomatique pour la spécification des paramètres et la gestion des résultats.

Consultez [AWS Tools for Windows PowerShell](#) pour démarrer.

Toolkit for VS Code

Le [AWS Toolkit for Visual Studio Code](#) est un plugin pour l'éditeur de code Visual Studio (VS Code). La boîte à outils facilite le développement, le débogage et le déploiement d'applications qui utilisent AWS.

Avec la boîte à outils, vous pouvez faire notamment les choses suivantes :

- Créez des applications sans serveur contenant des AWS Lambda fonctions, puis déployez les applications sur une AWS CloudFormation pile.
- Travaillez avec les EventBridge schémas Amazon.
- IntelliSense À utiliser lorsque vous travaillez avec des fichiers de définition de tâches Amazon ECS.
- Visualisez une AWS Cloud Development Kit (AWS CDK) application.

Toolkit for Visual Studio

Il AWS Toolkit for Visual Studio s'agit d'un plugin pour l'IDE Visual Studio qui facilite le développement, le débogage et le déploiement d'applications .NET utilisant Amazon Web Services.

Le Toolkit for Visual Studio fournit des modèles Visual Studio pour des services tels que Lambda et des assistants de déploiement pour les applications Web et les applications sans serveur. Vous pouvez utiliser l' AWS explorateur pour gérer les EC2 instances Amazon, utiliser les tables Amazon DynamoDB, publier des messages dans les files d'attente Amazon Simple Notification Service (Amazon SNS), etc., le tout dans Visual Studio.

Pour commencer, voir [Configuration du AWS Toolkit for Visual Studio](#).

Boîte à outils pour Azure DevOps

AWS Toolkit pour Microsoft Azure DevOps Ajoute des tâches pour permettre facilement aux pipelines de création et de publication dans Azure DevOps et Azure DevOps Server de fonctionner avec AWS les services. Vous pouvez travailler avec Amazon S3,, AWS Elastic Beanstalk AWS CodeDeploy, Lambda AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) et Amazon SNS. Vous pouvez également exécuter des commandes à l'aide du PowerShell module Windows et du AWS Command Line Interface (AWS CLI).

Pour commencer à utiliser le AWS Toolkit pour Azure DevOps, consultez le [guide de AWS Toolkit pour Microsoft Azure DevOps l'utilisateur](#).

Plateformes soutenues par le AWS SDK pour .NET

AWS SDK pour .NET Il fournit des groupes distincts d'assemblages permettant aux développeurs de cibler différentes plateformes. Toutefois, toutes les fonctionnalités SDK ne sont pas identiques sur toutes ces plateformes. Cette rubrique aborde les différences de prise en charge pour chaque plateforme.

.NET Core

Il AWS SDK pour .NET prend en charge les applications écrites pour .NET Core (.NET Core 3.1, .NET 5, .NET 6, etc.). AWS les clients de service prennent uniquement en charge les modèles d'appel asynchrones dans .NET Core. Cela affecte également de nombreuses abstractions de haut niveau créées à partir de clients de service, comme Amazon S3TransferUtility, qui ne prend en charge que les appels asynchrones dans l'environnement .NET Core.

.NET Standard 2.0

Les variantes non liées au framework AWS SDK pour .NET sont conformes au [.NET Standard 2.0](#). AWS SDK pour .NET fournit uniquement des méthodes asynchrones pour les applications écrites selon le standard .NET.

.NET Framework 4.5

Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

Cette version AWS SDK pour .NET est compilée avec .NET Framework 4.5 et s'exécute dans le runtime .NET 4.0. AWS [les clients de service prennent en charge les modèles d'appel synchrones et asynchrones et utilisent les mots clés async et await introduits dans C# 5.0](#).

.NET Framework 3.5

Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

Cette version du AWS SDK pour .NET est compilée avec .NET Framework 3.5 et s'exécute dans le runtime .NET 2.0 ou .NET 4.0. AWS les clients de service prennent en charge les modèles d'appel synchrones et asynchrones et utilisent l'ancien modèle de début et de fin.

Note

AWS SDK pour .NET Il n'est pas conforme à la norme fédérale de traitement de l'information (FIPS) lorsqu'il est utilisé par des applications conçues selon la version 2.0 du CLR. Pour plus de détails sur la façon dont vous pouvez remplacer une implémentation conforme à la norme

FIPS dans cet environnement, consultez [CryptoConfig](#) blog Microsoft et la HMACSHA256 classe (HMACSHA256Cng) de l'équipe de [sécurité CLR](#) dans Security.Cryptography.dll.

Bibliothèque de classes portable et Xamarin

AWS SDK pour .NET Il contient également une implémentation de bibliothèque de classes portable. L'implémentation de la bibliothèque de classes portable peut cibler plusieurs plateformes, notamment Universal Windows Platform (UWP) et Xamarin sur iOS et Android. Consultez le [SDK mobile pour .NET et Xamarin and Xamarin pour](#) plus de détails. AWS les clients de service ne prennent en charge que les modèles d'appel asynchrones.

Support Unity

Pour plus d'informations sur le support Unity, consultez [Considérations spéciales relatives au support Unity](#).

En savoir plus

[Migration vers la version 3.5 du AWS SDK pour .NET](#)

AWS SDKs Guide de référence et d'outils

Le [guide de référence sur les outils AWS SDKs et](#) contient des informations pertinentes et importantes pour de nombreuses boîtes à outils AWS SDKs et les AWS CLI. Voici quelques exemples des informations contenues dans la référence :

- Informations sur les [credentials fichiers partagés AWSconfig et](#) leur [emplacement](#).
- [Configuration AWS des comptes, des utilisateurs et des rôles](#)
- [Référence des paramètres de configuration et d'authentification](#)
- [AWS bibliothèques CRT \(Common Runtime\)](#)
- [AWS SDKs et politique de maintenance des outils](#)
- [AWS SDKs et matrice de support des versions d'outils](#)

Historique des révisions

Pour découvrir ce qui a changé dans les différentes versions, consultez ce qui suit :

- [Journaux des modifications apportées au SDK](#)
- [Quoi de neuf dans le AWS SDK pour .NET](#)
- [Historique du document](#)

Quoi de neuf dans le AWS SDK pour .NET

Pour obtenir des informations de haut niveau sur les nouveaux développements liés à la, AWS SDK pour .NET consultez la page du produit sur <https://aws.amazon.com/sdk-for-net/> et les [journaux des modifications du SDK](#).

Voici les nouveautés du AWS SDK pour .NET.

17 octobre 2025 : version 2.0 de l'outil de AWS déploiement

La version 2.0 de l'outil de AWS déploiement pour .NET CLI a été publiée. Pour plus d'informations sur l'outil de déploiement, consultez [Déployez des applications pour AWS](#). Pour plus d'informations sur la version 2.0 de l'outil de déploiement, consultez le billet de blog [Nouveautés de l'outil de AWS déploiement pour .NET](#).

21 août 2025 : Prochainement end-of-support pour la version 3 du AWS SDK pour .NET

La version end-of-support pour cette version (V3) du AWS SDK pour .NET a été annoncée. Consultez le [guide de migration vers la version 4](#) pour migrer vos applications V3 et éviter les interruptions. Pour plus d'informations, consultez le billet de blog [Announcing the end-of-support for the AWS SDK pour .NET v3](#).

5 mai 2025 : Disponibilité générale du cadre de traitement des AWS messages pour .NET

Le [cadre de traitement des AWS messages pour .NET](#) est généralement disponible ! Le framework est un framework AWS natif qui simplifie le développement d'applications de traitement de messages .NET utilisant des AWS services tels qu'Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) et Amazon. EventBridge

28 avril 2025 : Version 4 du AWS SDK pour .NET

La version 4 du AWS SDK pour .NET est généralement disponible ! Pour plus d'informations, consultez le [guide du développeur AWS SDK pour .NET \(V4\)](#), en particulier la rubrique relative à la [migration vers la version 4](#).

15 février 2025 : Intégrations avec .NET Aspire

Des intégrations avec .NET Aspire visant à améliorer la boucle de développement interne ont été publiées. Pour plus d'informations, consultez [Intégration AWS à .NET Aspire dans le AWS SDK pour .NET](#).

10 février 2025 : sortie de l'AG pour l'observabilité

L'observabilité est la mesure dans laquelle l'état actuel d'un système peut être déduit des données qu'il émet. L'observabilité a été ajoutée au AWS SDK pour .NET, y compris la mise en œuvre d'un fournisseur de télémétrie. Pour plus d'informations, consultez [Observabilité](#) ce guide et le billet de blog [Annonce de la disponibilité générale des OpenTelemetry bibliothèques AWS .NET](#).

15 janvier 2025 : Nouveau comportement par défaut pour la protection de l'intégrité

À partir de la version 3.7.412.0 du AWS SDK pour .NET, le SDK fournit des protections d'intégrité par défaut en calculant automatiquement une somme de contrôle pour les téléchargements. CRC32 Pour plus d'informations, voir l' GitHub annonce sur <https://github.com/aws/aws-sdk-net/issues/3610>. Le SDK fournit également des paramètres globaux pour les protections de l'intégrité des données que vous pouvez définir en externe. Vous pouvez en savoir plus sur les [protections de l'intégrité des données](#) dans le [guide de référence sur les outils AWS SDKs et](#).

15 novembre 2024 : article de blog pour la version 4, sortie de la version 4

La version 4 de la AWS SDK pour .NET version 4 a été publiée le 15 novembre 2024. Pour plus d'informations sur cet aperçu, consultez le billet de blog [Aperçu 4 de la version 4 de la AWS SDK pour .NET version 4](#).

16 août 2024 : article de blog pour la version préliminaire 1, sortie de la version 4

La AWS SDK pour .NET version 4 a été publiée en tant que premier aperçu le 16 août 2024. Pour plus d'informations sur cet aperçu, consultez le billet de blog [Aperçu 1 de la version AWS SDK pour .NET 4](#).

23 février 2024 : Ajout du support pour .NET 8

Support pour .NET 8 a été ajouté au AWS SDK pour .NET. Utilisez les derniers [NuGet packages](#) ou [assemblies compatibles avec .NET 8 et versions ultérieures](#). Vous trouverez des informations supplémentaires sur ce support, notamment sur le [support de Lambda](#), dans le billet de blog [.NET 8 Support](#) sur AWS.

18 février 2024 : modifications à venir concernant le support du .NET Framework

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

17/07/2023 : Le framework d' AWS Lambda annotations a été publié pour une mise à disposition générale

Le [framework AWS Lambda Annotations](#) rend l'expérience d'écriture de fonctions Lambda en C# plus naturelle pour les développeurs .NET en utilisant la technologie de génération de sources C#. Il est désormais disponible pour tous.

15/07/2023 : Le fournisseur de cache distribué pour DynamoDB a été publié en version préliminaire

La bibliothèque du fournisseur de cache distribué permet d'utiliser Amazon DynamoDB comme espace de stockage pour le framework de cache distribué d'ASP.NET Core. [Pour plus d'informations, consultez le billet de blog Présentation du fournisseur de cache distribué AWS .NET pour DynamoDB \(version préliminaire\) et du référentiel. GitHub](#)

07-13 : L'outil de AWS déploiement a été publié

L'outil de AWS déploiement a été publié. Il s'agit d'un outil interactif destiné à la CLI .NET AWS Toolkit for Visual Studio qui permet de déployer des applications .NET avec un minimum de AWS connaissances et avec le moins de clics ou de commandes. Pour de plus amples informations, veuillez consulter [Déployez des applications pour AWS](#).

24/08/2020 : La version 3.5 du SDK est sortie

- Standardisation de l'expérience .NET en transférant la prise en charge de toutes les variantes non liées au framework du SDK vers .NET Standard 2.0. Pour plus d'informations, consultez [Migration vers la version 3.5](#).
- Ajout de paginateurs à de nombreux clients de service, ce qui facilite la pagination des résultats de l'API. Pour de plus amples informations, veuillez consulter [Paginateurs](#).

Ressources supplémentaires

Services pris en charge

Il AWS SDK pour .NET prend en charge la plupart des produits d' AWS infrastructure, et de nouveaux services sont ajoutés fréquemment. Pour obtenir la liste des AWS services pris en charge par le SDK, consultez le [fichier README du SDK](#).

Page d'accueil du AWS SDK pour .NET

Pour plus d'informations sur le AWS SDK pour .NET, consultez la page d'accueil du SDK à <https://aws.amazon.com/sdk-for-net/> l'adresse.

Documentation de référence du SDK

La documentation de référence du SDK vous permet de parcourir et de rechercher l'ensemble du code inclus dans le SDK. Il fournit une documentation complète et des exemples d'utilisation. Pour plus d'informations, consultez la page [Référence de l'API AWS SDK pour .NET](#).

AWS Re:post (anciennement AWS forums)

Consultez [AWS RE:Post](#), en particulier le [sujet concerné AWS SDK pour .NET, pour poser des questions ou faire part de vos commentaires](#). AWS Chaque page de documentation contient un lien Try AWS Re:post en bas de page qui vous amène à la rubrique Re:Post associée. AWS les ingénieurs surveillent les sujets et répondent aux questions, aux commentaires et aux problèmes.

Si vous êtes connecté à Re:post, vous pouvez également suivre un sujet. Pour suivre le sujet du AWS SDK pour .NET, rendez-vous sur la [page Tous les sujets](#), recherchez « .NET activé AWS », puis cliquez sur le bouton Suivre.

Trousses à outils

- AWS Toolkit for Visual Studio: Si vous utilisez l'IDE Microsoft Visual Studio, vous devez consulter le [guide de l'AWS Toolkit for Visual Studio utilisateur](#).
- AWS Toolkit for Visual Studio Code: Si vous utilisez l'IDE Microsoft Visual Studio, vous devez consulter le [guide de l'AWS Toolkit for Visual Studio Code utilisateur](#).

Bibliothèques, extensions et outils utiles

Consultez les aws-sdk-net référentiels [aws/dotnet](#) et [aws/](#) du GitHub site Web pour obtenir des liens vers des bibliothèques, des outils et des ressources que vous pouvez utiliser pour créer des applications et des services .NET sur lesquels vous pouvez créer des applications et des services .NET. AWS

Voici quelques exemples :

- [AWS Extension de configuration .NET pour Systems Manager](#)
- [AWS Configuration des extensions .NET Core](#)
- [AWS Journalisation .NET](#)
- [Amazon Cognito Authentication Extension Library](#)
- [Kit SDK AWS X-Ray pour .NET](#)

Autres ressources

Voici d'autres ressources qui pourraient s'avérer utiles :

- [Réseau des développeurs](#)
- [Environnement de développement .NET sur le AWS cloud - Déploiement de référence Quick Start](#)
- [Bonjour, Cloud ! blogue](#)
- AWS Livre blanc : [Développement et déploiement d'applications .NET](#) sur AWS
- [AWS Microservice Extractor for .NET](#)
- [Assistant de portage pour .NET](#)
- [AWS SDKs Guide de référence et d'outils](#)

Commencez avec le AWS SDK pour .NET

Pour utiliser le AWS SDK pour .NET, vous devez installer votre chaîne d'outils et configurer un certain nombre d'éléments essentiels dont votre application a besoin pour accéder aux AWS services. Il s'agit des licences suivantes :

- Un compte ou un rôle d'utilisateur approprié
- Informations d'authentification pour ce compte utilisateur ou pour assumer ce rôle
- Spécification de la AWS région
- AWSSDK packages ou assemblages

Certaines des rubriques de cette section fournissent des informations sur la façon de configurer ces éléments essentiels.

Les autres rubriques de cette section et d'autres sections fournissent des informations sur des méthodes plus avancées de configuration de votre projet.

Rubriques

- [Installez et configurez votre chaîne d'outils](#)
- [Configurer l'authentification du SDK avec AWS](#)
- [Faites une visite rapide du AWS SDK pour .NET](#)
- [Lancement d'un nouveau projet](#)
- [Configuration de la AWS région](#)
- [Installez AWSSDK des packages avec NuGet](#)
- [Installez AWSSDK des assemblages sans NuGet](#)
- [Résolution des informations d'identification et des profils](#)
- [Informations supplémentaires sur les utilisateurs et les rôles](#)
- [Configuration avancée pour votre AWS SDK pour .NET projet](#)
- [Utilisation d'informations d'identification existantes](#)

Installez et configurez votre chaîne d'outils

Pour utiliser le AWS SDK pour .NET, vous devez avoir installé certains outils de développement.

Développement multiplateforme

Les éléments suivants sont requis pour le développement .NET multiplateforme sous Windows, Linux ou macOS :

- Microsoft [.NET Core SDK](#), version 2.1, 3.1 ou ultérieure, qui inclut l'interface de ligne de commande (CLI) NET (**dotnet**) et le moteur d'exécution .NET core.
- Un éditeur de code ou un environnement de développement intégré (IDE) adapté à votre système d'exploitation et à vos exigences. Il s'agit généralement d'une solution qui fournit un certain support pour .NET Core.

Les exemples incluent [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) et [Microsoft Visual Studio](#).

- (Facultatif) Une AWS boîte à outils, si elle est disponible, pour l'éditeur que vous avez choisi et pour votre système d'exploitation.

Les exemples incluent le [AWS Toolkit for Visual Studio Code](#), [AWS Toolkit for JetBrains](#), et [AWS Toolkit for Visual Studio](#).

Windows avec Visual Studio et .NET Core

Les éléments suivants sont requis pour le développement sous Windows avec Visual Studio et .NET Core :

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1, 3.1 ou version ultérieure

Ceci est généralement inclus par défaut lors de l'installation d'une version récente de Visual Studio.

- (Facultatif) Le AWS Toolkit for Visual Studio, qui est un plugin qui fournit une interface utilisateur pour gérer vos AWS ressources et vos profils locaux à partir de Visual Studio. Pour installer le kit d'outils, voir [Configuration du AWS Toolkit for Visual Studio](#).

Pour plus d'informations, consultez le [AWS Toolkit for Visual Studio Guide de l'utilisateur](#).

Étape suivante

[Configurer l'authentification du SDK avec AWS](#)

Configurer l'authentification du SDK avec AWS

Vous devez définir la manière dont votre code s'authentifie AWS lorsque vous développez avec Services AWS. Vous pouvez configurer l'accès programmatique aux AWS ressources de différentes manières, en fonction de l'environnement et de l' AWS accès dont vous disposez.

Pour découvrir les différentes méthodes d'authentification pour le SDK, consultez la section [Authentification et accès](#) dans le guide de référence des outils AWS SDKs et.

Cette rubrique part du principe qu'un nouvel utilisateur se développe localement, qu'il n'a pas reçu de méthode d'authentification de la part de son employeur et qu'il l'utilisera AWS IAM Identity Center pour obtenir des informations d'identification temporaires. Si votre environnement s'écarte de ces hypothèses, certaines des informations contenues dans cette rubrique ne s'appliquent peut-être pas à votre cas ou vous ont déjà été communiquées.

La configuration de cet environnement nécessite plusieurs étapes, qui sont résumées comme suit :

1. [Connectez-vous en utilisant les informations d'identification de console](#)
2. [Activation et configuration d'IAM Identity Center](#)
3. [Configurez le SDK pour utiliser IAM Identity Center.](#)
4. [Démarrer une session sur le portail AWS d'accès](#)

Connectez-vous en utilisant les informations d'identification de console

Vous pouvez utiliser vos identifiants de connexion existants à la console de AWS gestion pour accéder aux AWS services par programmation. Après un flux d'authentification basé sur un navigateur, AWS génère des informations d'identification temporaires qui fonctionnent avec les outils de développement locaux tels que la AWS CLI Outils AWS pour PowerShell et le AWS SDK pour .NET v3.

Grâce à ce processus, vous pouvez vous authentifier à l'aide des informations d'identification root créées lors de la configuration initiale du compte, d'un utilisateur IAM ou d'une identité fédérée auprès

de votre fournisseur d'identité, et la AWS CLI gère Outils AWS pour PowerShell automatiquement les informations d'identification temporaires pour vous. Cette approche améliore la sécurité en éliminant le besoin de stocker des informations d'identification à long terme localement.

Lorsque vous exécutez la `aws login` commande depuis la AWS CLI, vous pouvez effectuer une sélection parmi vos sessions de console actives ou vous connecter via le flux d'authentification basé sur le navigateur, ce qui générera automatiquement des informations d'identification temporaires. Le jeton d'identification temporaire expirera dans 15 minutes, mais le AWS SDK pour .NET v3 actualise automatiquement le jeton lorsque cela est nécessaire lors de vos demandes.

 Warning

Cette fonctionnalité n'est pas prise en charge pour les cibles .NET Framework.

 Important

Si vous utilisez les informations d'identification de la console pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution AWS de connexion puisse fonctionner :

- AWSSDK3. Se connecter

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

- [Connectez-vous à l'aide des informations d'identification de la console via la AWS CLI.](#)

Le config fichier ressemblera à ceci une fois que vous serez connecté en suivant les étapes indiquées ci-dessus dans la AWS CLI.

```
[default]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1

[my-dev-profile]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1
```

Activation et configuration d'IAM Identity Center

Pour utiliser IAM Identity Center, il doit d'abord être activé et configuré. Pour en savoir plus sur la procédure à suivre pour le SDK, consultez l'étape 1 de la rubrique consacrée à l'[authentification IAM Identity Center](#) dans le guide de référence AWS SDKs and Tools. Suivez particulièrement toutes les instructions nécessaires dans I do not have established access through IAM Identity Center.

Configurez le SDK pour utiliser IAM Identity Center.

Les informations relatives à la configuration du SDK pour utiliser IAM Identity Center se trouvent à l'étape 2 de la rubrique consacrée à l'[authentification IAM Identity Center](#) dans le guide de référence AWS SDKs and Tools. Une fois cette configuration terminée, votre système doit contenir les éléments suivants :

- Le AWS CLI, que vous utilisez pour démarrer une session de portail d' AWS accès avant d'exécuter votre application.
- Le AWS config fichier partagé qui contient un [\[default\]profil](#) avec un ensemble de valeurs de configuration pouvant être référencées à partir du SDK. Pour connaître l'emplacement de ce fichier, reportez-vous à la section [Emplacement des fichiers partagés](#) dans le Guide de référence des outils AWS SDKs et. AWS SDK pour .NET Utilise le fournisseur de jetons SSO du profil pour obtenir des informations d'identification avant d'envoyer des demandes à AWS. La valeur `sso_role_name`, qui est un rôle IAM connecté à un ensemble d'autorisations IAM Identity Center, doit autoriser l'accès aux Services AWS utilisés dans votre application.

L'exemple de fichier config suivant illustre un profil par défaut configuré avec le fournisseur de jetons SSO. Le paramètre `sso_session` du profil fait référence à la section `sso-session` nommée. La `sso-session` section contient les paramètres permettant de lancer une session sur le portail AWS d'accès.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
```

```
sso_registration_scopes = sso:account:access
```

⚠ Important

Si vous l'utilisez AWS IAM Identity Center pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Démarrer une session sur le portail AWS d'accès

Avant d'exécuter une application qui y accède Services AWS, vous avez besoin d'une session de portail d' AWS accès active pour que le SDK utilise l'authentification IAM Identity Center pour résoudre les informations d'identification. En fonction de la durée de session que vous avez configurée, votre accès finira par expirer et le SDK rencontrera une erreur d'authentification. Pour vous connecter au portail AWS d'accès, exécutez la commande suivante dans le AWS CLI.

```
aws sso login
```

Étant donné que vous disposez d'une configuration de profil par défaut, il n'est pas nécessaire d'appeler la commande avec l'option `--profile`. Si la configuration de votre fournisseur de jetons SSO utilise un profil nommé, la commande est `aws sso login --profile named-profile`.

Pour vérifier si vous avez déjà une session active, exécutez la AWS CLI commande suivante.

```
aws sts get-caller-identity
```

La réponse à cette commande doit indiquer le compte IAM Identity Center et l'ensemble d'autorisations configurés dans le fichier partagé `config`.

 Note

Si vous disposez déjà d'une session active sur le portail AWS d'accès et que vous l'exécutez `aws sso login`, il ne vous sera pas demandé de fournir d'informations d'identification.

Le processus de connexion peut vous demander d'autoriser l' AWS CLI accès à vos données. Comme AWS CLI il repose sur le SDK pour Python, les messages d'autorisation peuvent contenir des variantes du `botocore` nom.

Informations supplémentaires

- Pour plus d'informations sur l'utilisation d'IAM Identity Center et du SSO dans un environnement de développement, consultez [Authentification unique](#) la [Authentification avancée](#) section. Ces informations incluent des méthodes alternatives et plus avancées, ainsi que des didacticiels qui vous montrent comment utiliser ces méthodes.
- Pour plus d'options d'authentification pour le SDK, telles que l'utilisation de profils et de variables d'environnement, consultez le chapitre de [configuration](#) du guide de référence des outils AWS SDKs et des outils.
- Pour en savoir plus sur les bonnes pratiques, consultez [Bonnes pratiques de sécurité dans IAM](#) dans le Guide de l'utilisateur IAM.
- Pour créer des AWS informations d'identification à court terme, consultez la section [Informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.
- Pour en savoir plus sur les autres fournisseurs d'informations d'identification, consultez la section [Fournisseurs d'informations d'identification standardisés](#) dans le guide de référence sur les outils AWS SDKs et.

Faites une visite rapide du AWS SDK pour .NET

Cette section fournit des didacticiels de base pour les développeurs qui découvrent le AWS SDK pour .NET.

 Note

Avant d'utiliser ces didacticiels, vous devez d'abord avoir [installé votre chaîne d'outils](#) et [configuré l'authentification du SDK](#).

Pour plus d'informations sur le développement de logiciels pour des AWS services spécifiques ainsi que pour des exemples de code, consultez [Travaillez avec les AWS services](#). Pour des exemples de code supplémentaires, voir [SDK pour .NET exemples de code](#).

Rubriques

- [Application multiplateforme simple utilisant AWS SDK pour .NET](#)
- [Application Windows simple utilisant le AWS SDK pour .NET](#)
- [Étapes suivantes](#)

Application multiplateforme simple utilisant AWS SDK pour .NET

Ce didacticiel utilise .NET Core AWS SDK pour .NET et .NET pour le développement multiplateforme. Le didacticiel explique comment utiliser le SDK pour répertorier les [compartiments Amazon S3](#) que vous possédez et, éventuellement, créer un compartiment.

Vous allez réaliser ce didacticiel à l'aide d'outils multiplateformes comme l'interface de ligne de commande (CLI) .NET. Pour d'autres méthodes de configuration de votre environnement de développement, consultez [Installez et configurez votre chaîne d'outils](#).

Nécessaire pour le développement .NET multiplateforme sous Windows, Linux ou macOS :

- Microsoft [.NET Core SDK](#), version 2.1, 3.1 ou ultérieure, qui inclut l'interface de ligne de commande (CLI) NET (**dotnet**) et le moteur d'exécution .NET core.
- Un éditeur de code ou un environnement de développement intégré (IDE) adapté à votre système d'exploitation et à vos exigences. Il s'agit généralement d'une solution qui fournit un certain support pour .NET Core.

Les exemples incluent [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) et [Microsoft Visual Studio](#).

 Note

Avant d'utiliser ces didacticiels, vous devez d'abord avoir [installé votre chaîne d'outils](#) et [configuré l'authentification du SDK](#).

Étapes

- [Création du projet](#)
- [Créer le code](#)
- [Exécutez l'application](#)
- [Nettoyage](#)

Création du projet

1. Ouvrez l'invite de commande ou le terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

```
dotnet new console --name S3CreateAndList
```

3. Accédez au S3CreateAndList dossier que vous venez de créer et exécutez les commandes suivantes :

```
dotnet add package AWSSDK.S3  
dotnet add package AWSSDK.SecurityToken  
dotnet add package AWSSDK.SSO  
dotnet add package AWSSDK.SSO0IDC
```

Les commandes précédentes installent les NuGet packages depuis le [gestionnaire de NuGet packages](#). Comme nous savons exactement de quels NuGet packages nous avons besoin pour ce didacticiel, nous pouvons effectuer cette étape dès maintenant. Il est également courant que les packages requis soient connus au cours du développement. Dans ce cas, une commande similaire peut être exécutée à ce moment-là.

Créer le code

1. Dans le dossier S3CreateAndList, recherchez et ouvrez Program.cs dans votre éditeur de code.
2. Remplacez le contenu par le code suivant et enregistrez le fichier.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSO0IDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
        //   privileges for
        //   STS and S3 buckets.
        // - An active SSO Token.
        //   If an active SSO token isn't available, the SSO user should do the
        //   following:
        //   In a terminal, the SSO user must call "aws sso login".

        // Class members.
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
```

```

        Console.WriteLine($"\\nSSO Profile:\\n {await
ssoProfileClient.GetCallerIdentityArn()}");

        // Create the S3 client is by using the SSO credentials obtained
earlier.
        var s3Client = new AmazonS3Client(ssoCreds);

        // Parse the command line arguments for the bucket name.
        if (GetBucketName(args, out String bucketName))
        {
            // If a bucket name was supplied, create the bucket.
            // Call the API method directly
            try
            {
                Console.WriteLine($"\\nCreating bucket {bucketName}...");
                var createResponse = await s3Client.PutBucketAsync(bucketName);
                Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
            }
            catch (Exception e)
            {
                Console.WriteLine("Caught exception when creating a bucket:");
                Console.WriteLine(e.Message);
            }
        }

        // Display a list of the account's S3 buckets.
        Console.WriteLine("\\nGetting a list of your buckets...");
        var listResponse = await s3Client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
        foreach (S3Bucket b in listResponse.Buckets)
        {
            Console.WriteLine(b.BucketName);
        }
        Console.WriteLine();
    }

    //
    // Method to parse the command line.
    private static Boolean GetBucketName(string[] args, out String bucketName)
    {
        Boolean retval = false;
        bucketName = String.Empty;
        if (args.Length == 0)

```

```

        {
            Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
                "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
            bucketName = String.Empty;
            retval = false;
        }
        else if (args.Length == 1)
        {
            bucketName = args[0];
            retval = true;
        }
        else
        {
            Console.WriteLine("\nToo many arguments specified." +
                "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
                "\n\nUsage: S3CreateAndList [bucket_name]" +
                "\n - bucket_name: A valid, globally unique bucket name." +
                "\n - If bucket_name isn't supplied, this utility simply lists
your buckets.");
            Environment.Exit(1);
        }
        return retval;
    }

    //
    // Method to get SSO credentials from the information in the shared config
    file.
    static AWSCredentials LoadSsoCredentials(string profile)
    {
        var chain = new CredentialProfileStoreChain();
        if (!chain.TryGetAWSCredentials(profile, out var credentials))
            throw new Exception($"Failed to find the {profile} profile");
        return credentials;
    }
}

// Class to read the caller's identity.
public static class Extensions
{
    public static async Task<string> GetCallerIdentityArn(this
IAmazonSecurityTokenService stsClient)

```

```
{
    var response = await stsClient.GetCallerIdentityAsync(new
    GetCallerIdentityRequest());
    return response.Arn;
}
}
```

Exécutez l'application

1. Exécutez la commande suivante.

```
dotnet run
```

2. Examinez le résultat pour connaître le nombre de compartiments Amazon S3 que vous possédez, le cas échéant, ainsi que leurs noms.
3. Choisissez un nom pour un nouveau compartiment Amazon S3. Utilisez « dotnet-quicktour-s 3-1-cross- » comme base et ajoutez-y quelque chose d'unique, comme un GUID ou votre nom. Assurez-vous de suivre les règles relatives aux noms de compartiments, telles que décrites dans la section [Règles relatives à la dénomination des compartiments](#) dans le [guide de l'utilisateur Amazon S3](#).
4. Exécutez la commande suivante en la *amzn-s3-demo-bucket* remplaçant par le nom du bucket que vous avez choisi.

```
dotnet run amzn-s3-demo-bucket
```

5. Examinez le résultat pour voir le nouveau compartiment créé.

Nettoyage

Au cours de ce didacticiel, vous avez créé des ressources que vous pouvez choisir de nettoyer pour le moment.

- Si vous ne souhaitez pas conserver le compartiment créé par l'application lors d'une étape précédente, supprimez-le à l'aide de la console Amazon S3 à l'adresse <https://console.aws.amazon.com/s3/>.
- Si vous ne souhaitez pas conserver votre projet .NET, supprimez le dossier `S3CreateAndList` de votre environnement de développement.

Étapes suivantes

Retournez au [menu de visite rapide](#) ou allez directement à la [fin de cette visite rapide](#).

Application Windows simple utilisant le AWS SDK pour .NET

Ce didacticiel utilise AWS SDK pour .NET Windows avec Visual Studio et .NET Core. Le didacticiel explique comment utiliser le SDK pour répertorier les [compartiments Amazon S3](#) que vous possédez et éventuellement créer un compartiment.

Vous exécuterez ce didacticiel sous Windows avec Visual Studio et .NET Core. Pour d'autres méthodes de configuration de votre environnement de développement, consultez [Installez et configurez votre chaîne d'outils](#).

Nécessaire pour le développement sous Windows avec Visual Studio et .NET Core :

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1, 3.1 ou version ultérieure

Ceci est généralement inclus par défaut lors de l'installation d'une version récente de Visual Studio.

Note

Avant d'utiliser ces didacticiels, vous devez d'abord avoir [installé votre chaîne d'outils](#) et [configuré l'authentification du SDK](#).

Étapes

- [Création du projet](#)
- [Créer le code](#)
- [Exécutez l'application](#)
- [Nettoyage](#)

Création du projet

1. Ouvrez Visual Studio et créez un nouveau projet qui utilise la version C# du modèle d'application console, c'est-à-dire avec la description suivante : «... pour créer une application de ligne de commande pouvant s'exécuter sur .NET... ». Appelez le projet `S3CreateAndList`.

Note

Ne choisissez pas la version .NET Framework du modèle d'application de console ou, si c'est le cas, veillez à utiliser .NET Framework 4.7.2 ou version ultérieure.

2. Une fois le projet nouvellement créé chargé, choisissez Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Recherchez les NuGet packages suivants et installez-les dans le projet :
`AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO`, et `AWSSDK.SSO0IDC`

Ce processus installe les NuGet packages à partir du [gestionnaire de NuGet packages](#). Comme nous savons exactement de quels NuGet packages nous avons besoin pour ce didacticiel, nous pouvons effectuer cette étape dès maintenant. Il est également courant que les packages requis soient connus au cours du développement. Dans ce cas, appliquez un processus similaire pour les installer à ce moment-là.

4. Si vous avez l'intention d'exécuter l'application à partir de l'invite de commande, ouvrez une invite de commande maintenant et accédez au dossier qui contiendra le résultat de la compilation. C'est généralement quelque chose comme `çaS3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, mais cela dépend de votre environnement.

Créer le code

1. Dans le projet `S3CreateAndList`, recherchez et ouvrez `Program.cs` dans l'IDE.
2. Remplacez le contenu par le code suivant et enregistrez le fichier.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSO0IDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
```

```
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-start.html
        // for complete steps.
        // Requirements:
        // - An SS0 profile in the SS0 user's shared config file with sufficient
        // privileges for
        // STS and S3 buckets.
        // - An active SS0 Token.
        // If an active SS0 token isn't available, the SS0 user should do the
        // following:
        // In a terminal, the SS0 user must call "aws sso login".

        // Class members.
        static async Task Main(string[] args)
        {
            // Get SS0 credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"SSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Create the S3 client is by using the SS0 credentials obtained
            // earlier.
            var s3Client = new AmazonS3Client(ssoCreds);

            // Parse the command line arguments for the bucket name.
            if (GetBucketName(args, out String bucketName))
            {
                // If a bucket name was supplied, create the bucket.
                // Call the API method directly
                try
                {
```



```
        Console.WriteLine($"\\nCreating bucket {bucketName}...");
        var createResponse = await s3Client.PutBucketAsync(bucketName);
        Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
    }
    catch (Exception e)
    {
        Console.WriteLine("Caught exception when creating a bucket:");
        Console.WriteLine(e.Message);
    }
}

// Display a list of the account's S3 buckets.
Console.WriteLine("\\nGetting a list of your buckets...");
var listResponse = await s3Client.ListBucketsAsync();
Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
foreach (S3Bucket b in listResponse.Buckets)
{
    Console.WriteLine(b.BucketName);
}
Console.WriteLine();
}

//
// Method to parse the command line.
private static Boolean GetBucketName(string[] args, out String bucketName)
{
    Boolean retval = false;
    bucketName = String.Empty;
    if (args.Length == 0)
    {
        Console.WriteLine("\\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
        "\\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
        bucketName = String.Empty;
        retval = false;
    }
    else if (args.Length == 1)
    {
        bucketName = args[0];
        retval = true;
    }
    else
```

```

        {
            Console.WriteLine("\nToo many arguments specified." +
                "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
                "\n\nUsage: S3CreateAndList [bucket_name]" +
                "\n - bucket_name: A valid, globally unique bucket name." +
                "\n - If bucket_name isn't supplied, this utility simply lists
your buckets.");
            Environment.Exit(1);
        }
        return retval;
    }

    //
    // Method to get SSO credentials from the information in the shared config
    file.
    static AWSCredentials LoadSsoCredentials(string profile)
    {
        var chain = new CredentialProfileStoreChain();
        if (!chain.TryGetAWSCredentials(profile, out var credentials))
            throw new Exception($"Failed to find the {profile} profile");
        return credentials;
    }
}

// Class to read the caller's identity.
public static class Extensions
{
    public static async Task<string> GetCallerIdentityArn(this
    IAmazonSecurityTokenService stsClient)
    {
        var response = await stsClient.GetCallerIdentityAsync(new
    GetCallerIdentityRequest());
        return response.Arn;
    }
}
}

```

3. Générer l'application

 Note

Si vous utilisez une ancienne version de Visual Studio, il est possible que vous obteniez une erreur de compilation similaire à la suivante :

« La fonctionnalité « `async main` » n'est pas disponible dans C# 7.0. Veuillez utiliser la version linguistique 7.1 ou supérieure. »

Si cette erreur s'affiche, configurez votre projet pour utiliser une version ultérieure de la langue. Cela se fait généralement dans les propriétés du projet, Build, Advanced.

Exécutez l'application

1. Exécutez l'application sans arguments de ligne de commande. Faites-le soit dans l'invite de commande (si vous en avez ouvert une plus tôt), soit depuis l'IDE.
2. Examinez le résultat pour connaître le nombre de compartiments Amazon S3 que vous possédez, le cas échéant, ainsi que leurs noms.
3. Choisissez un nom pour un nouveau compartiment Amazon S3. Utilisez « `dotnet-quicktour-s-3-1-winvs-` » comme base et ajoutez-y quelque chose d'unique, comme un GUID ou votre nom. Assurez-vous de suivre les règles relatives aux noms de compartiments, telles que décrites dans la section [Règles de dénomination](#) des compartiments du [guide de l'utilisateur Amazon S3](#).
4. Exécutez à nouveau l'application, en fournissant cette fois le nom du compartiment.

Dans la ligne de commande, remplacez *amzn-s3-demo-bucket* la commande suivante par le nom du bucket que vous avez choisi.

```
S3CreateAndList amzn-s3-demo-bucket
```

Ou, si vous exécutez l'application dans l'IDE, choisissez Project, S3 CreateAndList Properties, Debug et entrez le nom du bucket à cet endroit.

5. Examinez le résultat pour voir le nouveau compartiment créé.

Nettoyage

Au cours de ce didacticiel, vous avez créé des ressources que vous pouvez choisir de nettoyer pour le moment.

- Si vous ne souhaitez pas conserver le compartiment créé par l'application lors d'une étape précédente, supprimez-le à l'aide de la console Amazon S3 à l'adresse <https://console.aws.amazon.com/s3/>.
- Si vous ne souhaitez pas conserver votre projet .NET, supprimez le dossier S3CreateAndList de votre environnement de développement.

Étapes suivantes

Retournez au [menu de visite rapide](#) ou allez directement à la [fin de cette visite rapide](#).

Étapes suivantes

Veillez à nettoyer les ressources restantes que vous avez créées lors de l'exécution de ces didacticiels. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Maintenant que vous les avez visitées AWS SDK pour .NET, vous voudrez peut-être [commencer votre projet](#).

Lancement d'un nouveau projet

Il existe plusieurs techniques que vous pouvez utiliser pour démarrer un nouveau projet afin d'accéder aux AWS services. Voici certaines de ces techniques :

- Si vous débutez dans le développement .NET, AWS ou du moins si vous êtes novice dans le AWS SDK pour .NET domaine, vous pouvez consulter des exemples complets dans [Faites une visite rapide](#). Il vous présente le SDK.
- Vous pouvez démarrer un projet de base à l'aide de l'interface de ligne de commande .NET. Pour en voir un exemple, ouvrez une invite de commande ou un terminal, créez un dossier ou un répertoire, naviguez jusqu'à celui-ci, puis entrez ce qui suit.

```
dotnet new console --name [SOME-NAME]
```

Un projet vide est créé dans lequel vous pouvez ajouter du code et des NuGet packages. Pour plus d'informations, consultez le [guide .NET Core](#).

Pour consulter la liste des modèles de projets, utilisez ce qui suit : `dotnet new --list`

- AWS Toolkit for Visual Studio II inclut des modèles de projet C# pour une variété de AWS services. Après avoir [installé le kit d'outils](#) dans Visual Studio, vous pouvez accéder aux modèles lors de la création d'un nouveau projet.

Pour le voir, consultez la section [Utilisation des AWS services](#) dans le [guide de AWS Toolkit for Visual Studio l'utilisateur](#). Plusieurs des exemples présentés dans cette section créent de nouveaux projets.

- Si vous développez avec Visual Studio sous Windows mais sans le AWS Toolkit for Visual Studio, utilisez les techniques habituelles pour créer un nouveau projet.

Pour voir un exemple, ouvrez Visual Studio et choisissez Fichier, Nouveau, Projet. Recherchez « .net core » et choisissez la version C# du modèle d'application console (.NET Core) ou d'application WPF (.NET Core). Un projet vide est créé dans lequel vous pouvez ajouter du code et des NuGet packages.

Vous trouverez quelques exemples de la manière d'utiliser les AWS services dans [Exemples de code avec conseils](#).

Important

Si vous l'utilisez AWS IAM Identity Center pour l'authentification, votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Configuration de la AWS région

AWS Les régions vous permettent d'accéder à AWS des services qui résident physiquement dans une région géographique spécifique. Cela peut être utile pour la redondance et pour conserver vos données et vos applications en cours d'exécution près de l'endroit où vous et vos utilisateurs y accédez.

Pour consulter la liste actuelle de toutes les régions et points de terminaison pris en charge pour chaque AWS service, consultez la section [Points de terminaison et quotas du service](#) dans le. Références générales AWS Pour consulter la liste des points de terminaison régionaux existants, consultez la section Points de [terminaison AWS de service](#). Pour obtenir des informations détaillées sur les régions, voir [Spécifier les AWS régions que votre compte peut utiliser](#).

Vous pouvez créer un client AWS de service qui se rend dans une [région donnée](#). Vous pouvez également configurer votre application avec une région qui sera utilisée pour [tous les clients AWS du service](#). Ces deux cas sont expliqués ci-après.

Création d'un client de service avec une région particulière

Vous pouvez spécifier la région pour tous les clients du AWS service dans votre application. La définition de la région de cette manière a priorité sur tout paramètre global pour ce client de service en particulier.

Région existante

Cet exemple vous montre comment instancier un [EC2 client Amazon](#) dans une région existante. Il utilise des [RegionEndpoint](#) champs définis.

```
using (AmazonEC2Client ec2Client = new AmazonEC2Client(RegionEndpoint.USWest2))
{
    // Make a request to EC2 in the us-west-2 Region using ec2Client
}
```

Nouvelle région utilisant la RegionEndpoint classe

Cet exemple vous montre comment créer un nouveau point de terminaison de région en utilisant [RegionEndpoint.GetBySystemName](#).

```
var newRegion = RegionEndpoint.GetBySystemName("us-west-new");
using (var ec2Client = new AmazonEC2Client(newRegion))
{
    // Make a request to EC2 in the new Region using ec2Client
}
```

Nouvelle région utilisant la classe de configuration du client de service

Cet exemple montre comment utiliser la ServiceURL propriété de la classe de configuration du client de service pour spécifier la région ; dans ce cas, en utilisant la classe [Amazon EC2 Config](#).

Cette technique fonctionne même si le point de terminaison régional ne suit pas le modèle de point de terminaison régional normal.

```
var ec2ClientConfig = new AmazonEC2Config
{
    // Specify the endpoint explicitly
    ServiceURL = "https://ec2.us-west-new.amazonaws.com"
};

using (var ec2Client = new AmazonEC2Client(ec2ClientConfig))
{
    // Make a request to EC2 in the new Region using ec2Client
}
```

Spécifiez une région pour tous les clients du service

Il existe plusieurs manières de spécifier une région pour tous les clients de AWS service créés par votre application. Cette région est utilisée pour les clients de service qui ne sont pas créés avec une région en particulier.

Recherche AWS SDK pour .NET une valeur de région dans l'ordre suivant.

Profils

Définissez un profil que votre application ou le SDK a chargé. Pour de plus amples informations, veuillez consulter [Résolution des informations d'identification et des profils](#).

Variables d'environnement

Défini dans la variable d'AWS_REGION environnement.

Sous Linux ou macOS :

```
export AWS_REGION='us-west-2'
```

Sous Windows :

```
set AWS_REGION=us-west-2
```

 Note

Si vous définissez cette variable d'environnement pour l'ensemble du système (en utilisant `export` ou `setx`), elle affecte l'ensemble des boîtes à outils, SDKs et pas seulement le AWS SDK pour .NET.

AWSConfigs classe

Défini en tant que [AWSConfigs.AWSRegion](#) propriété.

```
AWSConfigs.AWSRegion = "us-west-2";  
using (var ec2Client = new AmazonEC2Client())  
{  
    // Make request to Amazon EC2 in us-west-2 Region using ec2Client  
}
```

Résolution régionale

Si aucune des méthodes décrites ci-dessus n'est utilisée pour spécifier un Région AWS, les AWS SDK pour .NET tentatives de recherche d'une région dans laquelle le client du AWS service pourra opérer.

Ordre de résolution régional

1. Fichiers de configuration de l'application tels que `app.config` et `web.config`.
2. Variables d'environnement (`AWS_REGION` et `AWS_DEFAULT_REGION`).
3. Un profil dont le nom est spécifié par une valeur dans `AWSConfigs.AWSProfileName`.
4. Un profil dont le nom est spécifié par la variable d'`AWS_PROFILE` environnement.
5. Le `[default]` profil.
6. Métadonnées de l' EC2 instance Amazon (si elle est exécutée sur une EC2 instance).

Si aucune région n'est trouvée, le SDK émet une exception indiquant qu'aucune région n'est configurée pour le client de AWS service.

Informations spéciales sur la région de Chine (Pékin)

Pour utiliser les services dans la région Chine (Beijing), vous devez posséder un compte et les informations d'identification qui sont spécifiques à la région Chine (Beijing). Les comptes et informations d'identification AWS des autres régions ne fonctionneront pas pour la région de Chine (Pékin). De même, les comptes et informations d'identification de la région de Chine (Pékin) ne fonctionneront pas pour AWS les autres régions. Pour plus d'informations sur les terminaux et les protocoles disponibles dans la région de Chine (Pékin), voir Points de [terminaison de la région de Pékin](#).

Informations spéciales sur les nouveaux AWS services

De nouveaux AWS services peuvent être lancés dans un premier temps dans quelques régions, puis pris en charge dans d'autres régions. Dans ces cas, il n'est pas nécessaire d'installer le dernier SDK pour accéder aux nouvelles régions pour ce service. Vous pouvez spécifier les régions récemment ajoutées par client ou globalement, comme indiqué précédemment.

Installez AWSSDK des packages avec NuGet

[NuGet](#) est un système de gestion de paquets pour la plate-forme .NET. Avec NuGet, vous pouvez installer les [AWSSDKpackages](#), ainsi que plusieurs autres extensions, sur votre projet. Pour plus d'informations, consultez le référentiel [aws/dotnet](#) sur le site Web. GitHub

NuGet possède toujours les versions les plus récentes des AWSSDK packages, ainsi que les versions précédentes. NuGet connaît les dépendances entre les packages et installe automatiquement tous les packages requis.

Warning

La liste des NuGet packages peut inclure un package nommé simplement AWSSDK « » (sans identifiant ajouté). N'installez PAS ce NuGet package ; il s'agit d'un ancien package qui ne doit pas être utilisé pour de nouveaux projets.

Les packages installés avec NuGet sont stockés avec votre projet plutôt que dans un emplacement central. Cela vous permet d'installer des versions d'assemblage spécifiques à une application donnée sans créer de problèmes de compatibilité pour les autres applications. Pour plus d'informations NuGet, consultez la [NuGet documentation](#).

 Note

Si vous ne pouvez pas ou n'êtes pas autorisé à télécharger et à installer des NuGet packages par projet, vous pouvez obtenir les AWSSDK assemblages et les stocker localement (ou sur site).

Si cela s'applique à vous et que vous n'avez pas encore obtenu les AWSSDK assemblages, consultez [Obtention d' AWSSDK assemblages](#). Pour savoir comment utiliser les assemblages stockés localement, consultez [Installez AWSSDK des assemblages sans NuGet](#).

Utilisation NuGet à partir de l'invite de commande ou du terminal

1. Accédez aux [AWSSDK packages concernés NuGet](#) et déterminez les packages dont vous avez besoin pour votre projet, par exemple [AWSSDK.S3](#).
2. Copiez la commande .NET CLI depuis la page Web de ce package, comme indiqué dans l'exemple suivant.

```
dotnet add package AWSSDK.S3 --version 3.3.110.19
```

3. Dans le répertoire de votre projet, exécutez cette commande .NET CLI. NuGet installe également toutes les dépendances, telles que [AWSSDK.Core](#).

 Note

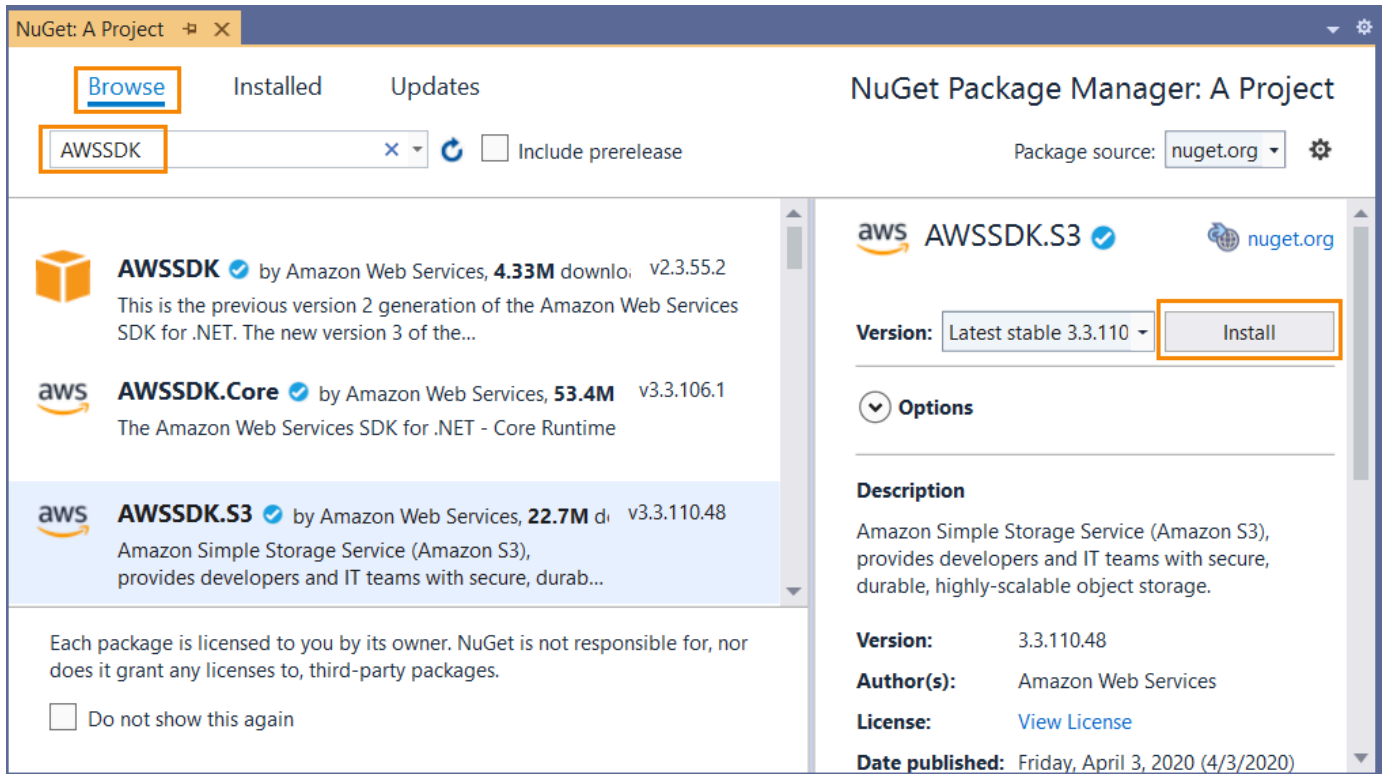
Si vous souhaitez uniquement disposer de la dernière version d'un NuGet package, vous pouvez exclure les informations de version de la commande, comme indiqué dans l'exemple suivant.

```
dotnet add package AWSSDK.S3
```

Utilisation NuGet depuis l'explorateur de solutions Visual Studio

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur votre projet, puis sélectionnez Gérer les NuGet packages dans le menu contextuel.
2. Dans le volet gauche du Gestionnaire de NuGet packages, choisissez Browse. Vous pouvez ensuite utiliser le champ de recherche pour rechercher le package que vous souhaitez installer. NuGet installe également toutes les dépendances, telles que [AWSSDK.Core](#).

La figure suivante montre l'installation du package AWSSDK.S3.



Utilisation NuGet depuis la console Package Manager

Dans Visual Studio, choisissez Outils, Gestionnaire de NuGet packages, Console du gestionnaire de packages.

Vous pouvez installer les AWSSDK packages souhaités à partir de la console Package Manager à l'aide de la **Install-Package** commande. Par exemple, pour installer [AWSSDK.S3](#), utilisez la commande suivante.

```
PM> Install-Package AWSSDK.S3
```

NuGet installe également toutes les dépendances, telles que [AWSSDK.Core](#).

Si vous devez installer une version antérieure d'un package, utilisez l'-Version option et spécifiez la version de package souhaitée, comme indiqué dans l'exemple suivant.

```
PM> Install-Package AWSSDK.S3 -Version 3.3.106.6
```

Pour plus d'informations sur les commandes de la console Package Manager, consultez la [PowerShellréférence](#) dans la [NuGetdocumentation](#) de Microsoft.

Installez AWSSDK des assemblages sans NuGet

Cette rubrique décrit comment utiliser les AWSSDK assemblages que vous avez obtenus et stockés localement (ou sur site) comme décrit dans [Obtention d' AWSSDK assemblages](#). Cette méthode n'est pas recommandée pour gérer les références au SDK, mais elle est requise dans certains environnements.

Note

La méthode recommandée pour gérer les références aux SDK consiste à télécharger et à installer uniquement les NuGet packages dont chaque projet a besoin. Cette méthode est décrite dans [Installez AWSSDK des packages avec NuGet](#).

Pour installer des AWSSDK assemblages

1. Créez un dossier dans votre zone de projet pour les AWSSDK assemblages requis. Par exemple, vous pouvez appeler ce dossier `AwsAssemblies`.
2. Si ce n'est pas déjà fait, [procurez-vous les AWSSDK assemblages](#), qui les placeront dans un dossier de téléchargement ou d'installation local. Copiez les fichiers DLL pour les assemblages requis depuis ce dossier de téléchargement dans votre projet (dans le `AwsAssemblies` dossier, dans notre exemple).

Veillez également à copier toutes les dépendances. Vous trouverez des informations sur les dépendances sur le [GitHub](#) site Web.

3. Faites référence aux assemblages requis comme suit.

Cross-platform development

1. Ouvrez le `.csproj` fichier de votre projet et ajoutez un `<ItemGroup>` élément.
2. Dans l'`<ItemGroup>` élément, ajoutez un `<Reference>` élément avec un `Include` attribut pour chaque assemblage requis.

Pour Amazon S3, par exemple, vous devez ajouter les lignes suivantes au `.csproj` fichier de votre projet.

Sous Linux et macOS :

```
<ItemGroup>
  <Reference Include="./AwsAssemblies/AWSSDK.Core.dll" />
  <Reference Include="./AwsAssemblies/AWSSDK.S3.dll" />
</ItemGroup>
```

Sous Windows :

```
<ItemGroup>
  <Reference Include="AwsAssemblies\AWSSDK.Core.dll" />
  <Reference Include="AwsAssemblies\AWSSDK.S3.dll" />
</ItemGroup>
```

3. Enregistrez le .csproj fichier de votre projet.

Windows with Visual Studio and .NET Core

1. Dans Visual Studio, chargez votre projet et ouvrez Projet, Ajouter une référence.
2. Cliquez sur le bouton Parcourir en bas de la boîte de dialogue. Accédez au dossier de votre projet et au sous-dossier dans lequel vous avez copié les fichiers DLL requis (AwsAssembliespar exemple).
3. Sélectionnez tous les fichiers DLL, choisissez Ajouter, puis OK.
4. Enregistrez votre projet.

Résolution des informations d'identification et des profils

Il AWS SDK pour .NET recherche les informations d'identification dans un certain ordre et utilise le premier ensemble disponible pour l'application en cours.

Ordre de recherche des informations d'identification

1. Informations d'identification définies explicitement sur le client AWS de service, comme décrit dans [Accès aux informations d'identification et aux profils dans une application](#).

Note

Cette rubrique figure dans la [Considérations spéciales](#) section car il ne s'agit pas de la méthode préférée pour spécifier les informations d'identification.

2. Un profil d'informations d'identification dont le nom est spécifié par une valeur dans [AWSConfigs](#). [AWSProfile](#)Nom.
3. Un profil d'informations d'identification dont le nom est spécifié par la variable d'AWS_PROFILEenvironnement.
4. Profil d'informations d'identification [default].
5. [AWSCredentialsSessions](#) créées à partir des variables d'AWS_SESSION_TOKENenvironnement AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY, et, si elles ne sont pas toutes vides.
6. [AWSCredentialsBasiques](#) créés à partir des variables d'AWS_SECRET_ACCESS_KEYenvironnement AWS_ACCESS_KEY_ID et, si elles ne sont pas toutes deux vides.
7. Le [fournisseur d'informations d'identification du conteneur](#).
8. Métadonnées de l' EC2 instance Amazon.

Si votre application s'exécute sur une EC2 instance Amazon, par exemple dans un environnement de production, utilisez un rôle IAM comme décrit dans [Octroi d'accès à l'aide d'un rôle IAM](#). Dans le cas contraire, comme lors des tests préliminaires, stockez vos informations d'identification dans un fichier utilisant le format de fichier AWS d'informations d'identification auquel votre application Web a accès sur le serveur.

Résolution du profil

Avec deux mécanismes de stockage différents pour les informations d'identification, il est important de comprendre comment les configurer AWS SDK pour .NET pour les utiliser. Le [AWSConfigs](#). [AWSProfiles](#)La [propriété de localisation](#) contrôle la manière dont AWS SDK pour .NET il trouve les profils d'identification.

AWSProfilesLocation	Comportement de résolution des profils
null (non défini) ou vide	Recherchez dans le SDK Store si la plateforme le prend en charge, puis recherchez le fichier

AWSProfilesLocation	Comportement de résolution des profils
	AWS d'informations d'identification partagé à l' emplacement par défaut . Si le profil ne se trouve dans aucun de ces emplacements, recherchez ~/.aws/config (Linux ou macOS) ou %USERPROFILE%\aws\config (Windows).
Le chemin d'accès à un fichier au format de fichier AWS d'informations d'identification	Recherche uniquement dans le fichier spécifié un profil portant le nom spécifié.

Utilisation des informations d'identification d'un compte utilisateur fédéré

Les applications qui utilisent le AWS SDK pour .NET ([AWSSDK.Core](#) version 3.1.6.0 et versions ultérieures) peuvent utiliser des comptes utilisateurs fédérés via Active Directory Federation Services (AD FS) pour accéder aux AWS services à l'aide du langage SAML (Security Assertion Markup Language).

La prise en charge de l'accès fédéré signifie que les utilisateurs peuvent s'authentifier en utilisant votre instance Active Directory. Des informations d'identification temporaires sont automatiquement accordées à l'utilisateur. Ces informations d'identification temporaires, valides pendant une heure, sont utilisées lorsque votre application invoque AWS des services. Le kit SDK se charge de gérer les informations d'identification temporaires. Pour les comptes utilisateur joints à un domaine, si votre application effectue un appel mais que les informations d'identification ont expiré, l'utilisateur est de nouveau authentifié automatiquement et de nouvelles informations d'identification lui sont octroyées. (Pour les non-domain-joined comptes, l'utilisateur est invité à saisir ses informations d'identification avant de se réauthentifier.)

Pour utiliser cette prise en charge dans votre application .NET, vous devez d'abord configurer le profil de rôle à l'aide d'une PowerShell applet de commande. Pour savoir comment procéder, consultez la [AWS Tools for Windows PowerShell documentation](#).

Après avoir configuré le profil de rôle, référez-le dans votre application. Il existe plusieurs façons de le faire, dont l'une consiste à utiliser le [AWSConfigs.AWSProfileNommez](#) la propriété de la même manière que vous le feriez avec les autres profils d'identification.

L'AWS Security Token Serviceassemblage ([AWSSDK. SecurityToken](#)) fournit le support SAML pour obtenir des AWS informations d'identification. Pour utiliser les informations d'identification d'un compte utilisateur fédéré, assurez-vous que cet assemblage est disponible pour votre application.

Spécifier des rôles ou des informations d'identification temporaires

Pour les applications qui s'exécutent sur EC2 des instances Amazon, le moyen le plus sûr de gérer les informations d'identification consiste à utiliser les rôles IAM, comme décrit dans[Octroi d'accès à l'aide d'un rôle IAM](#).

Pour les scénarios d'application dans lesquels le logiciel exécutable est accessible à des utilisateurs extérieurs à votre organisation, nous vous recommandons de concevoir le logiciel de manière à utiliser des informations d'identification de sécurité temporaires. En plus de fournir un accès restreint aux AWS ressources, ces informations d'identification ont l'avantage d'expirer après une période spécifiée. Pour plus d'informations sur les informations d'identification de sécurité temporaires, consultez les rubriques suivantes :

- [Informations d'identification de sécurité temporaires](#)
- [Groupes d'identités Amazon Cognito](#)

Utilisation des informations d'identification du proxy

Si votre logiciel communique avec AWS un proxy, vous pouvez spécifier les informations d'identification du proxy en utilisant la `ProxyCredentials` propriété de la `Config` classe d'un service. La `Config` classe d'un service fait généralement partie de l'espace de noms principal du service. Les exemples incluent les suivants : [AmazonCloudDirectoryConfig](#)sur [Amazon. CloudDirectory](#)espace de noms et [AmazonGameLiftConfig](#)dans [Amazon. GameLift](#)espace de noms.

Pour [Amazon S3](#), par exemple, vous pouvez utiliser un code similaire au suivant, où `SecurelyStoredUserName` et `SecurelyStoredPassword` sont le nom d'utilisateur et le mot de passe du proxy spécifiés dans un [NetworkCredential](#)objet.

```
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential(SecurelyStoredUserName,
    SecurelyStoredPassword);
```


 Note

Les versions antérieures du kit SDK utilisaient `ProxyUsername` et `ProxyPassword`, mais ces propriétés sont obsolètes.

Informations supplémentaires sur les utilisateurs et les rôles

Pour développer AWS ou exécuter des applications .NET sur .NET AWS, vous devez disposer d'une combinaison d'utilisateurs, d'ensembles d'autorisations et de rôles de service adaptés à ces tâches.

Les utilisateurs, les ensembles d'autorisations et les rôles de service spécifiques que vous créez, ainsi que la manière dont vous les utilisez, dépendent des exigences de vos applications. Vous trouverez ci-dessous des informations complémentaires indiquant pourquoi ils peuvent être utilisés et comment les créer.

Utilisateurs et ensembles d'autorisations

Bien qu'il soit possible d'accéder aux services AWS en utilisant un compte utilisateur IAM avec des informations d'identification à long terme, il ne s'agit plus d'une bonne pratique et est à éviter. Même pendant le développement, il est recommandé de créer des utilisateurs et des ensembles d'autorisations et d'utiliser AWS IAM Identity Center des informations d'identification temporaires fournies par une source d'identité.

Pour le développement, vous pouvez employer l'utilisateur que vous avez créé ou qui vous a été attribué dans [Configurer l'authentification du SDK](#). Si vous disposez des AWS Management Console autorisations appropriées, vous pouvez également créer différents ensembles d'autorisations avec le moins de privilèges pour cet utilisateur ou créer de nouveaux utilisateurs spécifiquement pour les projets de développement, en fournissant des ensembles d'autorisations dotés du moindre privilège. Le plan d'action que vous choisirez éventuellement dépend de votre situation.

Pour plus d'informations sur ces utilisateurs et ensembles d'autorisations et sur la façon de les créer, consultez [Authentification et accès](#) dans le guide de référence AWS SDKs et outils et [Démarrage](#) dans le guide de AWS IAM Identity Center l'utilisateur.

Rôles de service

Vous pouvez configurer un rôle AWS de service pour accéder aux AWS services pour le compte des utilisateurs. Ce type d'accès est approprié si plusieurs personnes doivent exécuter votre application à distance, par exemple sur une EC2 instance Amazon que vous avez créée à cette fin.

Le processus de création d'un rôle de service varie en fonction de la situation, mais il consiste principalement en ce qui suit.

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à <https://console.aws.amazon.com/iam/> l'adresse.
2. Cliquez sur Rôles, puis sur Créer un rôle.
3. Choisissez le AWS service, recherchez et sélectionnez EC2(par exemple), puis choisissez le cas EC2d'utilisation (par exemple).
4. Choisissez Suivant : Autorisations, puis sélectionnez les [politiques appropriées](#) pour les AWS services que votre application utilisera.

Warning

NE choisissez PAS cette AdministratorAccesspolitique, car elle autorise les autorisations de lecture et d'écriture pour presque tous les éléments de votre compte.

5. Choisissez Next : Tags et entrez les tags de votre choix.

Vous trouverez des informations sur les balises dans [Contrôler l'accès à l'aide de balises de AWS ressources](#) dans le [guide de l'utilisateur IAM](#).

6. Choisissez Suivant : Réviser et fournir un nom et une description du rôle. Puis choisissez Create role (Créer un rôle).

Vous trouverez des informations de haut niveau sur les rôles IAM dans [Identités \(utilisateurs, groupes et rôles\)](#) dans le guide de [l'utilisateur IAM](#). Vous trouverez des informations détaillées sur les rôles dans la rubrique [Rôles IAM](#) de ce guide.

Informations supplémentaires sur les rôles

- Utilisez des [rôles IAM](#) pour les tâches Amazon Elastic Container Service (Amazon ECS).
- Utilisez les [rôles IAM](#) pour les applications qui s'exécutent sur des EC2 instances Amazon.

Configuration avancée pour votre AWS SDK pour .NET projet

Les rubriques de cette section contiennent des informations sur les tâches et méthodes de configuration supplémentaires susceptibles de vous intéresser.

Rubriques

- [À l'aide de AWSSDK.Extensions.NETCore.Setup et interface IConfiguration](#)
- [Configuration d'autres paramètres de l'application](#)
- [Référence des fichiers de configuration pour AWS SDK pour .NET](#)

À l'aide de AWSSDK.Extensions.NETCore.Setup et interface IConfiguration

(Cette rubrique était auparavant intitulée « Configuration du AWS SDK pour .NET avec .NET Core »)

L'un des principaux changements apportés à .NET Core est la suppression des normes ConfigurationManager app.config et des web.config fichiers utilisés avec les applications .NET Framework et ASP.NET.

La configuration dans .NET Core est basée sur des paires clé-valeur établies par les fournisseurs de configuration. Les fournisseurs de configuration lisent les données de configuration en paires clé-valeur à partir de diverses sources de configuration, y compris des arguments de ligne de commande, des fichiers de répertoire, des variables d'environnement et des fichiers de paramètres.

Note

Pour plus d'informations, consultez [Configuration in ASP.NET Core](#).

Pour faciliter l'utilisation AWS SDK pour .NET avec .NET Core, vous pouvez utiliser les [AWSSDK.Extensions.NETCore NuGet .Package d'installation](#). Comme de nombreuses bibliothèques .NET Core, elle ajoute des méthodes d'extension à l'IConfigurationinterface pour faciliter la AWS configuration.

Le code source de ce package se trouve GitHub sur <https://github.com/aws/aws-sdk-net/tree/aws-sdk-net-v3.7/extensions/src/AWSSDK.Extensions.NETCore.Setup>.

À l'aide de AWSSDK.Extensions.NETCore.Configuration

Supposons que vous créiez une application ASP.NET Core Model-View-Controller (MVC), ce qui peut être réalisé à l'aide du modèle d'application Web ASP.NET Core dans Visual Studio ou en l'exécutant `dotnet new mvc ...` dans la CLI .NET Core. Lorsque vous créez une telle application, le constructeur de `Startup.cs` gère la configuration en lisant diverses sources d'entrée provenant de fournisseurs de configuration tels que `appsettings.json`.

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}
```

Pour utiliser l'`IConfiguration` objet afin d'obtenir les `AWSOptions`, ajoutez d'abord le `AWSSDK.Extensions.NETCore.Setup` NuGet package. Ajoutez ensuite vos options au fichier de configuration comme décrit ci-dessous.

Notez que l'un des fichiers ajoutés à votre projet est `appsettings.Development.json`. Cela correspond à un `EnvironmentName` ensemble de développement. Au cours du développement, vous insérez votre configuration dans ce fichier, qui n'est lu que lors des tests locaux. Lorsque vous déployez une instance Amazon EC2 `EnvironmentName` définie sur `Production`, ce fichier est ignoré et les informations d'identification IAM et la AWS SDK pour .NET région configurées pour l'instance Amazon EC2 sont renvoyées.

Les paramètres de configuration suivants présentent des exemples de valeurs que vous pouvez ajouter dans le `appsettings.Development.json` fichier de votre projet pour fournir des AWS paramètres.

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
  },
  "SupportEmail": "TechSupport@example.com"
}
```

Pour accéder à un paramètre dans un fichier CSHTML, utilisez la directive. `Configuration`

```
@using Microsoft.Extensions.Configuration
```

```
@inject IConfiguration Configuration

<h1>Contact</h1>

<p>
    <strong>Support:</strong> <a
    href='mailto:@Configuration["SupportEmail"]'>@Configuration["SupportEmail"]</a><br />
</p>
```

Pour accéder aux AWS options définies dans le fichier à partir du code, appelez la méthode d'`GetAWSSOptions` extension ajoutée à `IConfiguration`.

Pour construire un client de service à partir de ces options, appelez `CreateServiceClient`. L'exemple suivant montre comment créer un client de service Amazon S3. (Assurez-vous d'ajouter le NuGet package [AWSSDK.S3](#) à votre projet.)

```
var options = Configuration.GetAWSSOptions();
IAmazonS3 client = options.CreateServiceClient<IAmazonS3>();
```

Vous pouvez également créer plusieurs clients de service avec des paramètres incompatibles en utilisant plusieurs entrées dans le `appsettings.Development.json` fichier, comme indiqué dans les exemples suivants où la configuration pour `service1` inclut la `us-west-2` région et la configuration pour `service2` inclut l'URL du point de terminaison spécial.

```
{
  "service1": {
    "Profile": "default",
    "Region": "us-west-2"
  },
  "service2": {
    "Profile": "default",
    "ServiceURL": "URL"
  }
}
```

Vous pouvez ensuite obtenir les options pour un service spécifique en utilisant l'entrée dans le fichier JSON. Par exemple, pour obtenir les paramètres, `service1` utilisez ce qui suit.

```
var options = Configuration.GetAWSSOptions("service1");
```

Valeurs autorisées dans le fichier appsettings

Les valeurs de configuration d'application suivantes peuvent être définies dans le fichier `appsettings.Development.json`. Les noms des champs doivent utiliser le boîtier illustré. Pour plus de détails sur ces paramètres, consultez le [AWS.Runtime.ClientConfig](#) cours.

- Région
- Profil
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

Injection de dépendances ASP.NET Core

Les `AWSSDK.Extensions.NETCore` NuGet package `.Setup` s'intègre également à un nouveau système d'injection de dépendances dans ASP.NET Core. La `ConfigureServices` méthode de la `Startup` classe de votre application est celle où les services MVC sont ajoutés. Si l'application utilise Entity Framework, c'est également là que l'initialisation se produit.

```
public void ConfigureServices(IServiceCollection services)
```

```
{  
    // Add framework services.  
    services.AddMvc();  
}
```

Note

Des informations générales sur l'injection de dépendances dans .NET Core sont disponibles sur le [site de documentation .NET Core](#).

Le `AWSSDK.Extensions.NETCore.Setup` NuGet package ajoute de nouvelles méthodes d'extension `IServiceCollection` que vous pouvez utiliser pour ajouter AWS des services à l'injection de dépendances. Le code suivant explique comment ajouter les AWS options lues `IConfiguration` pour ajouter Amazon S3 et DynamoDB à la liste des services. (Assurez-vous d'ajouter les DBv2 NuGet packages [AWSSDK.S3](#) et [AWSSDK.Dynamo](#) à votre projet.)

```
public void ConfigureServices(IServiceCollection services)  
{  
    // Add framework services.  
    services.AddMvc();  
  
    services.AddDefaultAWSOptions(Configuration.GetAWSOptions());  
    services.AddAWSService<IAmazonS3>();  
    services.AddAWSService<IAmazonDynamoDB>();  
}
```

Désormais, si vos contrôleurs MVC utilisent `IAmazonS3` ou `IAmazonDynamoDB` en tant que paramètres dans leurs constructeurs, le système d'injection de dépendances transmet ces services.

```
public class HomeController : Controller  
{  
    IAmazonS3 S3Client { get; set; }  
  
    public HomeController(IAmazonS3 s3Client)  
    {  
        this.S3Client = s3Client;  
    }  
  
    ...  
}
```

```
}
```

Configuration d'autres paramètres de l'application

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework. Les Web.config fichiers App.config et ne sont pas présents par défaut dans les projets basés sur .NET Core.

Ouvrir pour afficher le contenu du .NET Framework

Vous pouvez configurer un certain nombre de paramètres d'application :

- [AWSLogging](#)
- [AWSLogMetrics](#)
- [AWSRegion](#)
- [AWSResponseLogging](#)
- [AWS.DynamoDBContext.TableNamePrefix](#)
- [AWS.S3.UseSignatureVersion4](#)
- [AWSEndpointDefinition](#)
- [AWS Points de terminaison générés par le service](#)

Ces paramètres peuvent être configurés dans le fichier App.config ou Web.config de l'application. Bien que vous puissiez également les configurer à l'aide de l' AWS SDK pour .NET API, nous vous recommandons d'utiliser le .config fichier de l'application. Les deux approches sont décrites ici.

Pour plus d'informations sur l'utilisation de l'<aws>élément, comme décrit plus loin dans cette rubrique, consultez la section [Référence des fichiers de configuration pour AWS SDK pour .NET](#).

AWSLogging

Configure comment SDK doit consigner les événements, le cas échéant. Par exemple, l'approche recommandée consiste à utiliser l'élément <logging>, élément enfant de l'élément <aws> :


```
<aws>
  <logging logTo="Log4Net"/>
</aws>
```

Autrement :

```
<add key="AWSLogging" value="log4net"/>
```

Les valeurs possibles sont :

None

Désactivez la journalisation des événements . Il s'agit de l'option par défaut.

log4net

Journaliser à l'aide de log4net.

SystemDiagnostics

Journaliser à l'aide de la classe System.Diagnostics.

Vous pouvez définir plusieurs valeurs pour l'attribut logTo, séparées par des virgules. L'exemple suivant définit la journalisation log4net et System.Diagnostics dans le fichier .config :

```
<logging logTo="Log4Net, SystemDiagnostics"/>
```

Autrement :

```
<add key="AWSLogging" value="log4net, SystemDiagnostics"/>
```

Vous pouvez également, à l'aide de l' AWS SDK pour .NET API, combiner les valeurs de l'[LoggingOptions](#) énumération et définir la propriété [AWSConfigs.Logging](#) :

```
AWSConfigs.Logging = LoggingOptions.Log4Net | LoggingOptions.SystemDiagnostics;
```

Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSLogMétriques

Spécifie si SDK doit ou non journaliser les métriques de performance. Pour définir la configuration de journalisation des métriques dans le fichier `.config`, définissez la valeur de l'attribut `logMetrics` dans l'élément `<logging>`, élément enfant de l'élément `<aws>` :

```
<aws>
  <logging logMetrics="true"/>
</aws>
```

Sinon, définissez la clé `AWSLogMetrics` dans la section `<appSettings>` :

```
<add key="AWSLogMetrics" value="true">
```

Sinon, pour définir la journalisation des métriques avec l' AWS SDK pour .NET API, définissez le [AWSConfigs.LogMetrics](#) propriété :

```
AWSConfigs.LogMetrics = true;
```

Ce paramètre configure la propriété `LogMetrics` par défaut pour tous les clients et toutes les configurations. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSRegion

Configure la AWS région par défaut pour les clients qui n'ont pas explicitement spécifié de région. Pour définir la région dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `region` dans l'élément `aws` :

```
<aws region="us-west-2"/>
```

Sinon, définissez la clé `AWSRegion` dans la section `<appSettings>` :

```
<add key="AWSRegion" value="us-west-2"/>
```

Sinon, pour définir la région à l'aide de l' AWS SDK pour .NET API, définissez le [AWSConfigs.AWSRegion](#) propriété :

```
AWSConfigs.AWSRegion = "us-west-2";
```

Pour plus d'informations sur la création d'un AWS client pour une région spécifique, consultez la section [Sélection de AWS région](#). Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes.

AWSResponseJournalisation

Configure quand SDK doit journaliser les réponses du service. Les valeurs possibles sont :

Never

Les réponses du service ne sont jamais journalisées. Il s'agit de l'option par défaut.

Always

Les réponses du service sont toujours journalisées.

OnError

Journaliser les réponses du service uniquement lorsqu'une erreur se produit.

Pour définir la configuration de journalisation de service dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `logResponses` dans l'élément `<logging>`, élément enfant de l'élément `<aws>` :

```
<aws>
  <logging logResponses="OnError"/>
</aws>
```

Vous pouvez également définir la clé de `AWSResponseJournalisation` dans la `<appSettings>` section :

```
<add key="AWSResponseLogging" value="OnError"/>
```

Sinon, pour configurer la journalisation des services avec l' AWS SDK pour .NET API, définissez le [AWSConfigs. ResponseLogging](#) propriété à l'une des valeurs de l'[ResponseLoggingOption](#) énumération :

```
AWSConfigs.ResponseLogging = ResponseLoggingOption.OnError;
```

Les modifications apportées à ce paramètre prennent effet immédiatement.

AWS.DynamoDBContext.TableNamePrefix

Configure la valeur par défaut de `TableNamePrefix` ; `DynamoDBContext` est utilisé si aucune configuration manuelle n'est effectuée.

Pour définir le préfixe du nom de table dans le fichier `.config`, l'approche recommandée consiste à définir la valeur de l'attribut `tableNamePrefix` dans l'élément `<dynamoDBContext>`, élément enfant de l'élément `<dynamoDB>`, lui-même élément enfant de l'élément `<aws>` :

```
<dynamoDBContext tableNamePrefix="Test-"/>
```

Sinon, définissez la clé `AWS.DynamoDBContext.TableNamePrefix` dans la section `<appSettings>` :

```
<add key="AWS.DynamoDBContext.TableNamePrefix" value="Test-"/>
```

Sinon, pour définir le préfixe du nom de table avec l' AWS SDK pour .NET API, définissez la propriété [AWSConfigs.DynamoDBContext.TableNamePrefix](#) :

```
AWSConfigs.DynamoDBContextTableNamePrefix = "Test-";
```

Les modifications apportées à ce paramètre prennent effet uniquement pour les instances nouvellement construites de `DynamoDBContextConfig` et `DynamoDBContext`.

AWS.S3.UseSignatureVersion4

Configure si le client Amazon S3 doit ou non utiliser la signature version 4 pour signer avec les demandes.

Pour définir la signature de la version 4 pour Amazon S3 dans le `.config` fichier, l'approche recommandée consiste à définir l'`useSignatureVersion4`attribut de l'`<s3>`élément, qui est un élément enfant de l'`<aws>`élément :

```
<aws>
  <s3 useSignatureVersion4="true"/>
</aws>
```

Vous pouvez également définir la `AWS.S3.UseSignatureVersion4` clé sur `true` dans la `<appSettings>` section :

```
<add key="AWS.S3.UseSignatureVersion4" value="true"/>
```

Sinon, pour définir la signature de la version 4 de la signature avec l' AWS SDK pour .NET API, définissez la propriété [AWSConfigs.S3 UseSignatureVersion 4](#) sur `true` :

```
AWSConfigs.S3UseSignatureVersion4 = true;
```

Par défaut, ce paramètre est `false`, mais Signature version 4 peut être utilisé par défaut dans certains cas ou avec certaines régions. Lorsque le paramètre est `true`, Signature version 4 est utilisé pour toutes les demandes. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances du client Amazon S3.

AWSEndpointDéfinition

Configure si SDK doit utiliser un fichier de configuration personnalisé qui définit les régions et les points de terminaison.

Pour définir le fichier de définition de point de terminaison dans le fichier `.config`, nous vous recommandons de définir la valeur de l'attribut `endpointDefinition` dans l'élément `<aws>`.

```
<aws endpointDefinition="c:\config\endpoints.json"/>
```

Vous pouvez également définir la clé de `AWSEndpointDéfinition` dans la `<appSettings>` section :

```
<add key="AWSEndpointDefinition" value="c:\config\endpoints.json"/>
```

Sinon, pour définir le fichier de définition du point de terminaison avec l' AWS SDK pour .NET API, définissez le [AWSConfigs. EndpointDefinition](#) propriété :

```
AWSConfigs.EndpointDefinition = @"c:\config\endpoints.json";
```

Si vous n'indiquez pas de nom de fichier, aucun fichier de configuration personnalisé ne sera utilisé. Les modifications apportées à ce paramètre ne prennent effet que pour les nouvelles instances AWS clientes. Le fichier `endpoint.json` est disponible sur <https://github.com/aws/aws-sdk-net/blob/aws-sdk-net-v3.7/sdk/src/Core/endpoints.json>

AWS Points de terminaison générés par le service

Certains AWS services génèrent leurs propres points de terminaison au lieu de consommer un point de terminaison régional. Les clients de ces services utilisent une URL de service spécifique à chaque service et à vos ressources. Amazon CloudSearch et AWS IoT. Les exemples suivants illustrent comment obtenir les points de terminaison pour ces services.

Exemple de CloudSearch point de terminaison Amazon

Le CloudSearch client Amazon est utilisé pour accéder au service CloudSearch de configuration Amazon. Vous utilisez le service CloudSearch de configuration Amazon pour créer, configurer et gérer des domaines de recherche. Pour créer un domaine de recherche, créez un [CreateDomainRequest](#) objet et indiquez la `DomainName` propriété. Créez un [AmazonCloudSearchClient](#) objet à l'aide de l'objet de demande. Appelez la méthode [CreateDomain](#). L'[CreateDomainResponse](#) objet renvoyé par l'appel contient une `DomainStatus` propriété qui possède à la fois les `SearchService` points de terminaison `DocService` et. Créez un [AmazonCloudSearchDomainConfig](#) objet et utilisez-le pour initialiser `DocService` les `SearchService` instances de la [AmazonCloudSearchDomainClient](#) classe.

```
// Create domain and retrieve DocService and SearchService endpoints
DomainStatus domainStatus;
using (var searchClient = new AmazonCloudSearchClient())
{
    var request = new CreateDomainRequest
    {
        DomainName = "testdomain"
    };
    domainStatus = searchClient.CreateDomain(request).DomainStatus;
    Console.WriteLine(domainStatus.DomainName + " created");
}

// Test the DocService endpoint
var docServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.DocService.Endpoint
};
using (var domainDocService = new AmazonCloudSearchDomainClient(docServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain DocService client instantiated using the DocService endpoint");
    Console.WriteLine("DocService endpoint = " + domainStatus.DocService.Endpoint);
}
```

```

    using (var docStream = new FileStream(@"C:\doc_source\XMLFile4.xml",
        FileMode.Open))
    {
        var upload = new UploadDocumentsRequest
        {
            ContentType = ContentType.ApplicationXml,
            Documents = docStream
        };
        domainDocService.UploadDocuments(upload);
    }
}

// Test the SearchService endpoint
var searchServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.SearchService.Endpoint
};
using (var domainSearchService = new
    AmazonCloudSearchDomainClient(searchServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain SearchService client instantiated using
        the SearchService endpoint");
    Console.WriteLine("SearchService endpoint = " +
        domainStatus.SearchService.Endpoint);

    var searchReq = new SearchRequest
    {
        Query = "Gambardella",
        Sort = "_score desc",
        QueryParser = QueryParser.Simple
    };
    var searchResp = domainSearchService.Search(searchReq);
}

```

AWS IoT Exemple de points de terminaison

Pour obtenir le point de terminaison pour AWS IoT, créez un [AmazonIoTClient](#) objet et appelez la [DescribeEndPoint](#) méthode. L'[DescribeEndPointResponse](#) objet renvoyé contient le `EndpointAddress`. Créez un [AmazonIoTDataConfig](#) objet, définissez la `ServiceURL` propriété et utilisez-le pour instancier la [AmazonIoTDataClient](#) classe.

```

string iotEndpointAddress;
using (var iotClient = new AmazonIoTClient())

```

```
{
    var endPointResponse = iotClient.DescribeEndpoint();
    iotEndpointAddress = endPointResponse.EndpointAddress;
}

var ioTdocServiceConfig = new AmazonIotDataConfig
{
    ServiceURL = "https://" + iotEndpointAddress
};
using (var dataClient = new AmazonIotDataClient(ioTdocServiceConfig))
{
    Console.WriteLine("AWS IoTData client instantiated using the endpoint from the
    IotClient");
}
```

Référence des fichiers de configuration pour AWS SDK pour .NET

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework. Les Web.config fichiers App.config et ne sont pas présents par défaut dans les projets basés sur .NET Core.

Ouvrir pour afficher le contenu du .NET Framework

Vous pouvez utiliser un projet App.config ou un Web.config fichier .NET pour spécifier des AWS paramètres, tels que les AWS informations d'identification, les options de journalisation, les points de terminaison de AWS service et AWS les régions, ainsi que certains paramètres pour AWS des services tels qu'Amazon DynamoDB EC2, Amazon et Amazon S3. Vous trouverez ci-dessous des informations expliquant comment mettre correctement en forme un fichier App.config ou Web.config pour spécifier ces types de paramètres.

Note

Bien que vous puissiez continuer à utiliser l'<appSettings>élément dans un Web.config fichier App.config OR pour spécifier des AWS paramètres, nous vous recommandons d'utiliser les <aws> éléments <configSections> et comme décrit plus loin dans cette

rubrique. Pour plus d'informations sur l'<appSettings>élément, consultez les exemples d'<appSettings>éléments dans [Configuration de votre AWS SDK pour .NET application](#).

Note

Bien que vous puissiez continuer à utiliser les propriétés de [AWSConfigs](#) classe suivantes dans un fichier de code pour spécifier des AWS paramètres, les propriétés suivantes sont obsolètes et ne seront peut-être pas prises en charge dans les versions futures :

- DynamoDBContextTableNamePrefix
- EC2UseSignatureVersion4
- LoggingOptions
- LogMetrics
- ResponseLoggingOption
- S3UseSignatureVersion4

En général, nous recommandons qu'au lieu d'utiliser les propriétés de AWSConfigs classe d'un fichier de code pour spécifier AWS les paramètres, vous utilisiez les <aws> éléments <configSections> et d'un Web.config fichier App.config or pour spécifier AWS les paramètres, comme décrit plus loin dans cette rubrique. Pour plus d'informations sur les propriétés précédentes, consultez les exemples de AWSConfigs code dans [Configuration de votre AWS SDK pour .NET application](#).

Rubriques

- [Déclarer une section AWS de paramètres](#)
- [Éléments autorisés](#)
- [Informations de référence sur les éléments](#)

Déclarer une section AWS de paramètres

Vous définissez AWS les paramètres dans un Web.config fichier App.config or à partir de l'<aws>élément. Avant de commencer à utiliser l'élément <aws>, vous devez créer un élément <section> (qui est un élément enfant de l'élément <configSections>) et affecter à son attribut

name la valeur `aws` et à son attribut `type` la valeur `Amazon.AWSSection, AWSSDK.Core`, comme dans l'exemple suivant :

```
<?xml version="1.0"?>
<configuration>
  ...
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <!-- Add your desired AWS settings declarations here. -->
  </aws>
  ...
</configuration>
```

L'éditeur Visual Studio ne fournit pas de saisie automatique du code (IntelliSense) pour l'<aws>élément ou ses éléments enfants.

Pour vous aider à créer une version correctement mise en forme de l'élément <aws>, appelez la méthode `Amazon.AWSConfigs.GenerateConfigTemplate`. Celle-ci génère une version canonique de l'élément <aws> sous la forme d'une chaîne bien formée, que vous pouvez adapter à vos besoins. Les sections suivantes décrivent les attributs et les éléments enfants de l'élément <aws>.

Éléments autorisés

Voici une liste des relations logiques entre les éléments autorisés dans une section de AWS paramètres. Vous pouvez générer la dernière version de cette liste en appelant la méthode `Amazon.AWSConfigs.GenerateConfigTemplate`, qui génère une version canonique de l'élément <aws> sous la forme d'une chaîne que vous pouvez adapter à vos besoins.

```
<aws
  endpointDefinition="string value"
  region="string value"
  profileName="string value"
  profilesLocation="string value">
  <logging
    logTo="None, Log4Net, SystemDiagnostics"
    logResponses="Never | OnError | Always"
    logMetrics="true | false"
    logMetricsFormat="Standard | JSON"
```

```

    logMetricsCustomFormatter="Namespace.Class, Assembly" />
<dynamoDB
  conversionSchema="V1 | V2">
    <dynamoDBContext
      tableNamePrefix="string value">
        <tableAliases>
          <alias
            fromTable="string value"
            toTable="string value" />
        </tableAliases>
        <map
          type="Namespace.Class, Assembly"
          targetTable="string value">
            <property
              name="string value"
              attribute="string value"
              ignore="true | false"
              version="true | false"
              converter="Namespace.Class, Assembly" />
          </map>
        </dynamoDBContext>
      </dynamoDB>
    <s3
      useSignatureVersion4="true | false" />
    <ec2
      useSignatureVersion4="true | false" />
    <proxy
      host="string value"
      port="1234"
      username="string value"
      password="string value" />
  </aws>

```

Informations de référence sur les éléments

Voici une liste des éléments autorisés dans une section de AWS paramètres. Pour chaque élément, ses attributs autorisés et éléments parents-enfants sont répertoriés.

Rubriques

- [alias](#)
- [aws](#)
- [dynamoDB](#)

- [dynamo DBContext](#)
- [ec2](#)
- [journalisation](#)
- [map](#)
- [property](#)
- [proxy](#)
- [s3](#)

alias

L'élément `<alias>` représente un élément unique dans une collection d'un ou plusieurs mappages entre la table de départ et la table d'arrivée qui spécifie une table différente de celle configurée pour un type. Cet élément est mappé à une instance de la classe `Amazon.Util.TableAlias` à partir de la propriété `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` du AWS SDK pour .NET. Le remappage intervient avant l'application d'un préfixe de nom de table.

Cet élément peut inclure les attributs suivants :

fromTable

Partie « table de départ » du mappage entre la table de départ et la table d'arrivée. Cet attribut est mappé à la propriété `Amazon.Util.TableAlias.FromTable` du AWS SDK pour .NET.

toTable

Partie « table d'arrivée » du mappage entre la table de départ et la table d'arrivée. Cet attribut est mappé à la propriété `Amazon.Util.TableAlias.ToTable` du AWS SDK pour .NET.

Le parent de l'élément `<alias>` est l'élément `<tableAliases>`.

L'élément `<alias>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<alias>` utilisé :

```
<alias
  fromTable="Studio"
  toTable="Studios" />
```

aws

L'<aws>élément représente l'élément le plus haut d'une section de AWS paramètres. Cet élément peut inclure les attributs suivants :

endpointDefinition

Le chemin absolu vers un fichier de configuration personnalisé qui définit les AWS régions et les points de terminaison à utiliser. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.EndpointDefinition` du AWS SDK pour .NET.

profileName

Le nom du profil pour les AWS informations d'identification stockées qui seront utilisées pour effectuer des appels de service. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSProfileName` du AWS SDK pour .NET.

profilesLocation

Le chemin absolu vers l'emplacement du fichier d'informations d'identification partagé avec d'autres AWS SDKs. Par défaut, le fichier d'informations d'identification est stocké dans le répertoire `.aws` situé dans le répertoire de base de l'utilisateur actif. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSProfilesLocation` du AWS SDK pour .NET.

region

L'ID de AWS région par défaut pour les clients qui n'ont pas explicitement spécifié de région. Cet attribut est mappé à la propriété `Amazon.AWSConfigs.AWSRegion` du AWS SDK pour .NET.

L'élément <aws> n'a pas d'élément parent.

L'élément <aws> peut inclure les éléments enfants suivants :

- <dynamoDB>
- <ec2>
- <logging>
- <proxy>
- <s3>

Voici un exemple de l'élément <aws> utilisé :

```
<aws
  endpointDefinition="C:\Configs\endpoints.xml"
  region="us-west-2"
  profileName="development"
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

L'élément `<dynamoDB>` représente un ensemble de paramètres pour Amazon DynamoDB. Cet élément peut inclure l'attribut `conversionSchema`, qui représente la version à utiliser pour la conversion entre des objets .NET et des objets DynamoDB. Les valeurs autorisées sont notamment V1 et V2. Cet attribut est mappé à la classe `Amazon.DynamoDBv2.DynamoDBEntryConversion` du AWS SDK pour .NET. Pour plus d'informations, consultez [DynamoDB Series – Conversion Schemas](#).

Le parent de l'élément `<dynamoDB>` est l'élément `<aws>`.

L'élément `<dynamoDB>` peut inclure l'élément enfant `<dynamoDBContext>`.

Voici un exemple de l'élément `<dynamoDB>` utilisé :

```
<dynamoDB
  conversionSchema="V2">
  <!-- ... -->
</dynamoDB>
```

dynamo DBContext

L'élément `<dynamoDBContext>` représente un ensemble de paramètres propres au contexte Amazon DynamoDB. Cet élément peut inclure l'attribut `tableNamePrefix`, qui représente le préfixe de nom de table par défaut que le contexte DynamoDB utilisera s'il n'est pas configuré manuellement. Cet attribut est mappé à la propriété `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` à partir de la propriété `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` du AWS SDK pour .NET. Pour plus d'informations, consultez [Enhancements to the DynamoDB SDK](#).

Le parent de l'élément `<dynamoDBContext>` est l'élément `<dynamoDB>`.

L'élément `<dynamoDBContext>` peut inclure les éléments enfants suivants :

- `<alias>` (une ou plusieurs instances)
- `<map>` (une ou plusieurs instances)

Voici un exemple de l'élément `<dynamoDBContext>` utilisé :

```
<dynamoDBContext
  tableNamePrefix="Test-">
  <!-- ... -->
</dynamoDBContext>
```

ec2

L'`<ec2>`élément représente un ensemble de EC2 paramètres Amazon. Cet élément peut inclure l'attribut `useSignatureVersion4`, qui indique si la signature version 4 sera utilisée pour toutes les demandes (vrai) ou si la signature version 4 ne sera pas utilisée pour toutes les demandes (faux, valeur par défaut). Cet attribut est mappé à la propriété `Amazon.Util.EC2Config.UseSignatureVersion4` à partir de la propriété `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` du AWS SDK pour .NET.

Le parent de l'`<ec2>`élément est l'élément.

L'élément `<ec2>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<ec2>` utilisé :

```
<ec2
  useSignatureVersion4="true" />
```

journalisation

L'élément `<logging>` représente un ensemble de paramètres pour la journalisation des réponses et la journalisation des métriques de performance. Cet élément peut inclure les attributs suivants :

logMetrics

Indique si les métriques de performance sont journalisées pour tous les clients et toutes les configurations (true) ; sinon, false. Cet attribut est mappé à la

propriété `Amazon.Util.LoggingConfig.LogMetrics` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetrics` du AWS SDK pour .NET.

logMetricsCustomFormatter

Type de données et nom d'assembly d'un formateur personnalisé pour les métriques de performance. Cet attribut est mappé à la propriété `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` du AWS SDK pour .NET.

logMetricsFormat

Format sous lequel se présentent les métriques de journalisation (mappé à la propriété `Amazon.Util.LoggingConfig.LogMetricsFormat` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` du AWS SDK pour .NET).

Les valeurs autorisées sont les suivantes :

JSON

Utilise le format JSON.

Standard

Utilise le format par défaut.

logResponses

Indique à quel moment les réponses du service sont journalisées (mappé à la propriété `Amazon.Util.LoggingConfig.LogResponses` à partir de la propriété `Amazon.AWSConfigs.LoggingConfig.LogResponses` du AWS SDK pour .NET).

Les valeurs autorisées sont les suivantes :

Always

Les réponses du service sont toujours journalisées.

Never

Les réponses du service ne sont jamais journalisées.

OnError

Les réponse du service sont journalisées uniquement en cas d'erreur.

logTo

Où se connecter (cartes de la LogTo propriété depuis la `Amazon.AWSConfigs.LoggingConfig.LogTo` propriété dans le AWS SDK pour .NET).

Les valeurs autorisées comprennent une ou plusieurs des valeurs suivantes :

Log4Net

Journalisation dans log4net.

None

Désactivation de la journalisation.

SystemDiagnostics

Journalisation dans `System.Diagnostics`.

Le parent de l'élément `<logging>` est l'élément `<aws>`.

L'élément `<logging>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<logging>` utilisé :

```
<logging
  logTo="SystemDiagnostics"
  logResponses="OnError"
  logMetrics="true"
  logMetricsFormat="JSON"
  logMetricsCustomFormatter="MyLib.Util.MyMetricsFormatter, MyLib" />
```

map

L'`<map>`élément représente un élément unique d'une collection de type-to-table mappages entre les types .NET et les tables DynamoDB (correspond à une instance de `TypeMapping` la classe à partir de la propriété `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` du). AWS SDK pour .NET Pour plus d'informations, consultez [Enhancements to the DynamoDB SDK](#).

Cet élément peut inclure les attributs suivants :

targetTable

Table DynamoDB à laquelle le mappage s'applique. Cet attribut est mappé à la propriété `Amazon.Util.TypeMapping.TargetTable` du AWS SDK pour .NET.

type

Type et nom d'assembly auxquels le mappage s'applique. Cet attribut est mappé à la propriété `Amazon.Util.TypeMapping.Type` du AWS SDK pour .NET.

Le parent de l'élément `<map>` est l'élément `<dynamoDBContext>`.

L'élément `<map>` peut inclure une ou plusieurs instances de l'élément enfant `<property>`.

Voici un exemple de l'élément `<map>` utilisé :

```
<map
  type="SampleApp.Models.Movie, SampleDLL"
  targetTable="Movies">
  <!-- ... -->
</map>
```

property

L'élément `<property>` représente une propriété DynamoDB. (Cet élément correspond à une instance d'`Amazon.Util.PropertyConfig` [classe issue de la `AddProperty` méthode décrite dans le AWS SDK pour .NET](#)) [Pour plus d'informations, consultez Améliorations apportées au SDK DynamoDB et aux attributs DynamoDB.](#)

Cet élément peut inclure les attributs suivants :

attribute

Nom d'un attribut de la propriété, par exemple le nom d'une clé de plage. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Attribute` du AWS SDK pour .NET.

converter

Type du convertisseur qui doit être utilisé pour cette propriété. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Converter` du AWS SDK pour .NET.

ignore

Indique si la propriété associée doit être ignorée (true) ; sinon, false. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Ignore` du AWS SDK pour .NET.

name

Le nom de la propriété. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Name` du AWS SDK pour .NET.

version

Indique si cette propriété doit stocker le numéro de version de l'élément (true) ; sinon, false. Cet attribut est mappé à la propriété `Amazon.Util.PropertyConfig.Version` du AWS SDK pour .NET.

Le parent de l'élément `<property>` est l'élément `<map>`.

L'élément `<property>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<property>` utilisé :

```
<property
  name="Rating"
  converter="SampleApp.Models.RatingConverter, SampleDLL" />
```

proxy

L'élément `<proxy>` représente les paramètres de configuration d'un proxy pour le AWS SDK pour .NET à utiliser. Cet élément peut inclure les attributs suivants :

hôte

Nom d'hôte ou adresse IP du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Host` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Host` du AWS SDK pour .NET.

mot de passe

Mot de passe permettant de s'authentifier auprès du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Password` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Password` du AWS SDK pour .NET.

port

Numéro de port du proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Port` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Port` du AWS SDK pour .NET.

nom d'utilisateur

Nom d'utilisateur permettant de s'authentifier auprès du serveur proxy. Cet attribut est mappé à la propriété `Amazon.Util.ProxyConfig.Username` à partir de la propriété `Amazon.AWSConfigs.ProxyConfig.Username` du AWS SDK pour .NET.

Le parent de l'élément `<proxy>` est l'élément `<aws>`.

L'élément `<proxy>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<proxy>` utilisé :

```
<proxy
  host="192.0.2.0"
  port="1234"
  username="My-Username-Here"
  password="My-Password-Here" />
```

s3

L'élément `<s3>` représente un ensemble de paramètres Amazon S3. Cet élément peut inclure l'attribut `useSignatureVersion4`, qui indique si la signature version 4 sera utilisée pour toutes les demandes (vrai) ou si la signature version 4 ne sera pas utilisée pour toutes les demandes (faux, valeur par défaut). Cet attribut est mappé à la propriété `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` du AWS SDK pour .NET.

Le parent de l'élément `<s3>` est l'élément `<aws>`.

L'élément `<s3>` ne contient aucun élément enfant.

Voici un exemple de l'élément `<s3>` utilisé :

```
<s3 useSignatureVersion4="true" />
```

Utilisation d'informations d'identification existantes

Les rubriques de cette section fournissent des informations sur l'utilisation d'informations d'identification à long terme ou à court terme sans utiliser AWS IAM Identity Center.

 Warning

Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que [AWS IAM Identity Center](#).

 Note

Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section [Autres méthodes d'authentification](#) dans le guide de référence des outils AWS SDKs et.

Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dans [Configurer l'authentification du SDK](#).

Avertissements et conseils importants concernant les informations d'identification

Avertissements concernant les informations d'identification

- N'utilisez PAS les informations d'identification root de votre compte pour accéder aux AWS ressources. Ces informations d'identification offrent un accès illimité au compte et sont difficiles à révoquer.
- N'incluez PAS de clés d'accès littérales ou d'informations d'identification dans vos fichiers d'application. Vous risqueriez en effet d'exposer accidentellement vos informations d'identification si, par exemple, vous chargez le projet sur un référentiel public.
- N'incluez PAS de fichiers contenant des informations d'identification dans votre zone de projet.
- Sachez que toutes les informations d'identification stockées dans le AWS `credentials` fichier partagé sont stockées en texte brut.

Conseils supplémentaires pour gérer les informations d'identification en toute sécurité

Pour une discussion générale sur la manière de gérer en toute sécurité les AWS informations d'identification, consultez AWS les [informations d'identification](#) de [sécurité dans le Références générales AWS](#) [guide de l'utilisateur IAM et les meilleures pratiques de sécurité et les cas d'utilisation](#).

En plus de ces exposés, envisagez de prendre les mesures suivantes :

- Créez des utilisateurs supplémentaires, tels que des utilisateurs dans IAM Identity Center, et utilisez leurs informations d'identification au lieu d'utiliser vos informations d'identification d'utilisateur root AWS . Les informations d'identification des autres utilisateurs peuvent être révoquées si nécessaire ou sont temporaires par nature. En outre, vous pouvez appliquer une politique à chaque utilisateur pour qu'il n'accède qu'à certaines ressources et actions et ainsi adopter les autorisations du moindre privilège.
- Utilisez des [rôles IAM](#) pour les tâches Amazon Elastic Container Service (Amazon ECS).
- Utilisez des [rôles IAM](#) pour les applications qui s'exécutent sur des instances Amazon EC2.
- Utilisez des [informations d'identification temporaires](#) ou des variables d'environnement pour les applications accessibles aux utilisateurs extérieurs à votre organisation.

Rubriques

- [Utilisation du fichier AWS d'informations d'identification partagé](#)
- [Utilisation du SDK Store \(Windows uniquement\)](#)

Utilisation du fichier AWS d'informations d'identification partagé

(N'oubliez pas de consulter les [avertissements et les instructions importants concernant les informations d'identification](#).)

L'un des moyens de fournir des informations d'identification pour vos applications consiste à créer des profils dans le fichier AWS d'informations d'identification partagé, puis à les stocker dans ces profils. Ce fichier peut être utilisé par l'autre AWS SDKs. Il peut également être utilisé par les [AWS CLI](#) [AWS Tools for Windows PowerShell](#), et les AWS boîtes à outils pour [Visual Studio](#) et [VS Code](#). [JetBrains](#)

⚠ Warning

Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que [AWS IAM Identity Center](#).

ℹ Note

Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section [Autres méthodes d'authentification](#) dans le guide de référence des outils AWS SDKs et.

Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dans [Configurer l'authentification du SDK](#).

Informations générales

Par défaut, le fichier AWS d'informations d'identification partagé se trouve dans le `.aws` répertoire de votre répertoire personnel et porte le nom `credentials` `~/.aws/credentials` (Linux ou macOS) ou `%USERPROFILE%\.aws\credentials` (Windows). Pour plus d'informations sur les emplacements alternatifs, consultez la section [Emplacement des fichiers partagés](#) dans le [Guide de référence des outils AWS SDKs et des outils](#). Voir aussi [Accès aux informations d'identification et aux profils dans une application](#).

Le fichier AWS d'informations d'identification partagé est un fichier en texte brut qui suit un certain format. Pour plus d'informations sur le format des fichiers AWS d'informations d'identification, consultez la section [Format du fichier d'informations d'identification](#) dans le guide de référence des outils AWS SDKs et.

Vous pouvez gérer les profils dans le fichier AWS d'informations d'identification partagé de plusieurs manières.

- Utilisez n'importe quel éditeur de texte pour créer et mettre à jour le fichier AWS d'informations d'identification partagé.

- Utilisez l'[Amazon.Runtime.CredentialManagement](#) espace de noms de l' AWS SDK pour .NET API, comme indiqué plus loin dans cette rubrique.
- Utilisez des commandes et des procédures pour [Outils AWS pour PowerShell](#) et les AWS boîtes à outils pour [Visual Studio](#) et [VS Code](#). [JetBrains](#)
- Utilisez [AWS CLI](#) des commandes, par exemple, `aws configure set aws_access_key_id etaws configure set aws_secret_access_key`.

Exemples de gestion de profils

Les sections suivantes présentent des exemples de profils dans le fichier AWS d'informations d'identification partagé. Certains exemples montrent le résultat, qui peut être obtenu par le biais de l'une des méthodes de gestion des informations d'identification décrites précédemment. D'autres exemples montrent comment utiliser une méthode particulière.

Le profil par défaut

Le fichier AWS d'informations d'identification partagé aura presque toujours un profil nommé default. C'est ici qu'ils AWS SDK pour .NET recherchent les informations d'identification si aucun autre profil n'est défini.

Le [default] profil ressemble généralement à ce qui suit.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Création d'un profil par programmation

Cet exemple montre comment créer un profil et l'enregistrer dans le fichier d' AWS informations d'identification partagé par programmation. Il utilise les classes suivantes de l'[Amazon.Runtime.CredentialManagement](#) namespace : [CredentialProfileOptions](#) [CredentialProfile](#), et. [SharedCredentialsFile](#)

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyID, SecurelyStoredSecretAccessKey);
...
```



```
void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var sharedFile = new SharedCredentialsFile();
    sharedFile.RegisterProfile(profile);
}
```

Warning

Un tel code ne devrait généralement pas figurer dans votre application. Si vous l'incluez dans votre application, prenez les précautions nécessaires pour vous assurer que les clés en texte brut ne soient pas visibles dans le code, sur le réseau ou même dans la mémoire de l'ordinateur.

Le profil créé par cet exemple est le suivant.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
```

Mettre à jour un profil existant par programmation

Cet exemple montre comment mettre à jour par programmation le profil créé précédemment. Il utilise les classes suivantes de l'[Amazon.Runtime.CredentialManagement](#) espace de noms : [CredentialProfile](#) et [SharedCredentialsFile](#). Il utilise également la [RegionEndpoint](#) classe de l'espace de noms [Amazon](#).

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...
```

```
void AddRegion(string profileName, RegionEndpoint region)
{
    var sharedFile = new SharedCredentialsFile();
    CredentialProfile profile;
    if (sharedFile.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        sharedFile.RegisterProfile(profile);
    }
}
```

Le profil mis à jour est le suivant.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
region=us-west-2
```

Note

Vous pouvez également définir la AWS région dans d'autres emplacements et en utilisant d'autres méthodes. Pour de plus amples informations, veuillez consulter [Configuration de la AWS région](#).

Utilisation du SDK Store (Windows uniquement)

(Assurez-vous de consulter les mises en [garde et les directives importantes](#).)

Sous Windows, le SDK Store est un autre endroit où vous pouvez créer des profils et stocker des informations d'identification chiffrées pour votre AWS SDK pour .NET application. Il est situé dans %USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json. Vous pouvez utiliser le SDK Store pendant le développement comme alternative au [fichier d'AWS informations d'identification partagé](#).

Warning

Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des

données réelles. Préférez la fédération avec un fournisseur d'identité tel que [AWS IAM Identity Center](#).

Note

Les informations contenues dans cette rubrique s'appliquent à des situations où vous devez obtenir et gérer manuellement des informations d'identification à court ou à long terme. Pour plus d'informations sur les informations d'identification à court et à long terme, consultez la section [Autres méthodes d'authentification](#) dans le guide de référence des outils AWS SDKs et.

Pour les meilleures pratiques de sécurité, utilisez AWS IAM Identity Center, comme décrit dans [Configurer l'authentification du SDK](#).

Informations générales

Le SDK Store offre les avantages suivants :

- Les informations d'identification du SDK Store sont cryptées et le SDK Store se trouve dans le répertoire personnel de l'utilisateur. Cela limite le risque d'exposition accidentelle de vos informations d'identification.
- Le SDK Store fournit également des informations d'identification aux [AWS Tools for Windows PowerShell](#) et [AWS Toolkit for Visual Studio](#).

Les profils SDK Store sont spécifiques à un utilisateur spécifique sur un hôte donné. Vous ne pouvez pas les copier vers d'autres hôtes ou d'autres utilisateurs. Cela signifie que vous ne pouvez pas réutiliser les profils SDK Store présents sur votre machine de développement pour d'autres hôtes ou machines de développement. Cela signifie également que vous ne pouvez pas utiliser les profils SDK Store dans les applications de production.

Vous pouvez gérer les profils dans le SDK Store de la manière suivante :

- Utilisez l'interface utilisateur graphique (GUI) du [AWS Toolkit for Visual Studio](#).
- Utilisez l'[Amazon.Runtime.CredentialManagement](#) espace de noms de l' AWS SDK pour .NET API, comme indiqué plus loin dans cette rubrique.

- Utilisez les commandes du [AWS Tools for Windows PowerShell](#); par exemple, `Set-AWSCredential` et `Remove-AWSCredentialProfile`.

Exemples de gestion de profils

Les exemples suivants vous montrent comment créer et mettre à jour par programmation un profil dans le SDK Store.

Création d'un profil par programmation

Cet exemple montre comment créer un profil et l'enregistrer dans le SDK Store par programmation. Il utilise les classes suivantes de l'[Amazon.Runtime.CredentialManagement](#) namespace : [CredentialProfileOptions](#), [CredentialProfile](#), et [Net SDKCredentials File](#).

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyId, SecurelyStoredSecretAccessKey);
...

void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var netSdkStore = new NetSDKCredentialsFile();
    netSdkStore.RegisterProfile(profile);
}
```

Warning

Un tel code ne devrait généralement pas figurer dans votre application. S'il est inclus dans votre application, prenez les précautions nécessaires pour vous assurer que les clés en texte brut ne soient pas visibles dans le code, sur le réseau ou même dans la mémoire de l'ordinateur.

Le profil créé par cet exemple est le suivant.

```
"[generated GUID]" : {
  "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
  "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
  "ProfileType" : "AWS",
  "DisplayName" : "my_new_profile",
}
```

Mettre à jour un profil existant par programmation

Cet exemple montre comment mettre à jour par programmation le profil créé précédemment. Il utilise les classes suivantes de l'[Amazon.Runtime.CredentialManagement](#) namespace : [CredentialProfile](#) et [Net SDK Credentials File](#). Il utilise également la [RegionEndpoint](#) classe de l'espace de noms [Amazon](#).

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...

void AddRegion(string profileName, RegionEndpoint region)
{
    var netSdkStore = new NetSDKCredentialsFile();
    CredentialProfile profile;
    if (netSdkStore.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        netSdkStore.RegisterProfile(profile);
    }
}
```

Le profil mis à jour est le suivant.

```
"[generated GUID]" : {
  "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
  "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
  "ProfileType" : "AWS",
  "DisplayName" : "my_new_profile",
  "Region" : "us-west-2"
}
```

 Note

Vous pouvez également définir la AWS région dans d'autres emplacements et en utilisant d'autres méthodes. Pour de plus amples informations, veuillez consulter [Configuration de la AWS région](#).

Caractéristiques du AWS SDK pour .NET

Cette section fournit des informations sur les fonctionnalités AWS SDK pour .NET que vous devrez peut-être prendre en compte lors de la création de vos applications.

Assurez-vous d'avoir d'abord [configuré votre projet](#).

Pour plus d'informations sur le développement de logiciels pour des AWS services spécifiques ainsi que des exemples de code, consultez [Travaillez avec les AWS services](#). Pour des exemples de code supplémentaires, voir [SDK pour .NET exemples de code](#).

Rubriques

- [AWS asynchrone APIs pour .NET](#)
- [Rétentatives et délais d'attente](#)
- [Paginateurs](#)
- [Observabilité](#)
- [Outils supplémentaires](#)

AWS asynchrone APIs pour .NET

AWS SDK pour .NET utilise le modèle asynchrone basé sur les tâches (TAP) pour son implémentation asynchrone. Pour en savoir plus sur le TAP, consultez la section [Modèle asynchrone basé sur les tâches \(TAP\)](#) sur docs.microsoft.com.

Cette rubrique vous donne un aperçu de la façon d'utiliser le TAP dans vos appels aux clients AWS du service.

Les méthodes asynchrones de l' AWS SDK pour .NET API sont des opérations basées sur la Task classe ou la Task<TResult> classe. [Consultez le site docs.microsoft.com pour obtenir des informations sur ces classes : classe Task, Task< > class. TResult](#)

Lorsque ces méthodes d'API sont appelées dans votre code, elles doivent être appelées dans une fonction déclarée avec le async mot clé, comme indiqué dans l'exemple suivant.

```
static async Task Main(string[] args)
```

```
{
    ...
    // Call the function that contains the asynchronous API method.
    // Could also call the asynchronous API method directly from Main
    // because Main is declared async
    var response = await ListBucketsAsync();
    Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
    ...
    var response = await s3Client.ListBucketsAsync();
    return response;
}
```

Comme indiqué dans l'extrait de code précédent, la portée préférée de la `async` déclaration est la `Main` fonction. La définition de cette `async` étendue garantit que tous les appels aux clients du AWS service doivent être asynchrones. Si vous ne pouvez pas `Main` déclarer que vous êtes asynchrone pour une raison quelconque, vous pouvez utiliser le `async` mot clé sur des fonctions autres que `Main` puis appeler les méthodes de l'API à partir de là, comme indiqué dans l'exemple suivant.

```
static void Main(string[] args)
{
    ...
    Task<ListBucketsResponse> response = ListBucketsAsync();
    Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
    ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
    ...
    var response = await s3Client.ListBucketsAsync();
    return response;
}
```

Notez la `Task<>` syntaxe spéciale requise `Main` lorsque vous utilisez ce modèle. En outre, vous devez utiliser le **`Result`** membre de la réponse pour obtenir les données.

Vous pouvez voir des exemples complets d'appels asynchrones à des clients AWS de service dans la [Faites une visite rapide](#) section ([Application multiplateforme simple](#) et [Application simple basée sur Windows](#)) et dans. [Exemples de code avec conseils](#)

Rétentatives et délais d'attente

Vous AWS SDK pour .NET permet de configurer le nombre de tentatives et les valeurs de délai d'expiration pour les requêtes HTTP adressées aux AWS services. Si les valeurs par défaut concernant les nouvelles tentatives et les délais d'expiration ne conviennent pas à votre application, vous pouvez les ajuster selon vos besoins spécifiques, mais il est important de comprendre en quoi le comportement de votre application en sera affecté.

Pour déterminer quelles valeurs utiliser pour les nouvelles tentatives et les délais d'expiration, prenez en compte les éléments suivants :

- Comment l'application AWS SDK pour .NET et votre application doivent-elles réagir lorsque la connectivité réseau se dégrade ou qu'un AWS service est inaccessible ? Souhaitez-vous que l'appel échoue rapidement ou est-il indiqué que l'appel continue de faire l'objet de nouvelles tentatives en votre nom ?
- Votre application est-elle une application ou un site web orienté vers l'utilisateur qui doit être réactive ou s'agit-il d'une tâche de traitement en arrière plan qui tolère davantage les latences accrues ?
- L'application est-elle déployée sur un réseau fiable à faible latence ou est-elle déployée sur un site distant avec une connectivité peu fiable ?

Nouvelles tentatives

Présentation de

Ils AWS SDK pour .NET peuvent réessayer les demandes qui échouent en raison d'un ralentissement côté serveur ou d'une interruption de connexion. Il existe deux propriétés des classes de configuration de service que vous pouvez utiliser pour spécifier le comportement d'un client de service en cas de nouvelle tentative. Les classes de configuration de service héritent de ces propriétés de l'abstrait [Amazon.Runtime.ClientConfig](#) classe de l'[AWS SDK pour .NET API Reference](#) :

- `RetryMode` spécifie l'un des trois modes de nouvelle tentative définis dans [Amazon.Runtime.RequestRetryMode](#) énumération.

La valeur par défaut de votre application peut être contrôlée à l'aide de la variable d'AWS_RETRY_MODE environnement ou du paramètre `retry_mode` dans le fichier de configuration partagé AWS .

- `MaxErrorRetry` indique le nombre de tentatives autorisées au niveau du client de service ; le SDK réessaie l'opération le nombre de fois spécifié avant d'échouer et de lancer une exception.

La valeur par défaut de votre application peut être contrôlée à l'aide de la variable d'AWS_MAX_ATTEMPTS environnement ou du paramètre `max_attempts` du fichier de AWS configuration partagé.

Les descriptions détaillées de ces propriétés se trouvent dans le résumé [Amazon.Runtime.ClientConfig](#) classe de l'[AWS SDK pour .NET API Reference](#). Chaque valeur de `RetryMode` correspond par défaut à une valeur particulière de `MaxErrorRetry`, comme indiqué dans le tableau suivant.

RetryMode	MaxErrorRetry Correspondant (Amazon DynamoDB)	MaxErrorRetry Correspondant (tous les autres)
Héritée	10	4
Standard	10	2
Adaptatif (expérimental)	10	2

Comportement

Quand votre candidature démarre

Lorsque votre application démarre, les valeurs par défaut pour `RetryMode` et `MaxErrorRetry` sont configurées par le SDK. Ces valeurs par défaut sont utilisées lorsque vous créez un client de service, sauf si vous spécifiez d'autres valeurs.

- Si les propriétés ne sont pas définies dans votre environnement, la valeur par défaut `RetryMode` est configurée comme `Legacy` et la valeur par défaut pour `MaxErrorRetry` est configurée avec la valeur correspondante du tableau précédent.
- Si le mode de nouvelle tentative a été défini dans votre environnement, cette valeur est utilisée par défaut pour `RetryMode`. La valeur par défaut pour `MaxErrorRetry` est configurée avec la valeur correspondante du tableau précédent, sauf si la valeur pour les erreurs maximales a également été définie dans votre environnement (décrit ci-dessous).
- Si la valeur du maximum d'erreurs a été définie dans votre environnement, cette valeur est utilisée par défaut pour `MaxErrorRetry`. Amazon DynamoDB fait exception à cette règle ; la valeur DynamoDB par défaut `MaxErrorRetry` pour est toujours celle du tableau précédent.

Pendant l'exécution de votre application

Lorsque vous créez un client de service, vous pouvez utiliser les valeurs par défaut pour `RetryMode` et `MaxErrorRetry`, comme décrit précédemment, ou vous pouvez spécifier d'autres valeurs. Pour spécifier d'autres valeurs, créez et incluez un objet de configuration de service tel qu'[AmazonDynamoDBConfig](#) ou [AmazonSQSConfig](#) lorsque vous créez le client de service.

Ces valeurs ne peuvent pas être modifiées pour un client de service après sa création.

Considérations

Lorsqu'une nouvelle tentative a lieu, la latence de votre demande augmente. Vous devez configurer les nouvelles tentatives en fonction des limites de votre application en termes de latence totale des demandes et de taux d'erreur.

Délais

Vous AWS SDK pour .NET permet de configurer les délais d'expiration des demandes au niveau du client de service et par appel de méthode. Il existe deux mécanismes pour configurer les délais d'expiration, qui sont abordés dans les sections suivantes :

- Si vous utilisez des [appels asynchrones](#), vous pouvez utiliser le `CancellationToken` paramètre de la méthode.
- Si vous utilisez des [appels synchrones dans .NET Framework](#), vous pouvez utiliser les `ReadWriteTimeout` propriétés `Timeout` et du fichier abstrait [Amazon.Runtime.ClientConfig](#) classe.

Utilisation du **CancellationToken** paramètre pour les délais

Vous AWS SDK pour .NET permet de configurer les délais d'expiration des demandes pour les appels asynchrones à l'aide du paramètre. **CancellationToken** L'extrait de code suivant montre un exemple. Le code est lancé **System.Threading.Tasks.TaskCanceledException** si la demande n'est pas terminée dans les 10 secondes.

```
string bucketName = "amzn-s3-demo-bucket";
string path = "pathToBucket";
using (var amazonS3Client = new AmazonS3Client(new AmazonS3Config()))
{
    // Cancel request after 10 seconds
    CancellationTokenSource cancellationTokenSource = new
    CancellationTokenSource(TimeSpan.FromMilliseconds(10000));
    CancellationToken cancellationToken = cancellationTokenSource.Token;
    ListObjectsV2Request listRequestV2 = new()
    {
        BucketName = bucketName,
        Prefix = path,
    };

    ListObjectsV2Response listResponseV2 = await
    amazonS3Client.ListObjectsV2Async(listRequestV2, cancellationToken);
}
```

Utilisation des **ReadWriteTimeout** propriétés **Timeout** et pour les délais d'expiration

Note

La **Timeout** propriété n'a aucun impact sur les appels asynchrones. Si vous utilisez des appels asynchrones, consultez [Utilisation du **CancellationToken** paramètre pour les délais](#) plutôt.

Vous AWS SDK pour .NET permet de configurer les valeurs du délai d'expiration de la demande et du read/write délai d'expiration du socket au niveau du client de service. Ces valeurs sont spécifiées dans les **ReadWriteTimeout** propriétés **Timeout** et du résumé [Amazon.Runtime.ClientConfig](#) classe. Ces valeurs sont transmises sous forme de **ReadWriteTimeout** propriétés **Timeout** et des [HttpWebRequest](#) objets créés par l'objet client du AWS service. La valeur par défaut de **Timeout** est de 100 secondes et celle de **ReadWriteTimeout** est de 300 secondes.

Lorsque votre réseau présente une latence élevée ou qu'une opération fait l'objet d'une nouvelle tentative du fait de certaines conditions, un long délai d'expiration et un nombre élevé de nouvelles tentatives peuvent donner l'impression que certaines opérations du kit SDK ne répondent pas.

Note

La version AWS SDK pour .NET qui cible la bibliothèque de classes portable (PCL) utilise la [HttpClient](#) classe au lieu de la [HttpWebRequest](#) classe et ne prend en charge que la propriété [Timeout](#).

Vous trouverez ci-dessous les exceptions aux valeurs de délai d'expiration par défaut. Ces valeurs sont remplacées lorsque vous définissez explicitement les valeurs de délai d'expiration.

- `Timeout` et `ReadWriteTimeout` sont définis sur les valeurs maximales si la méthode appelée télécharge un flux, tel que [AmazonS3Client.PutObjectAsync\(\)](#), client [Amazon S3.UploadPartAsync\(\)](#), [AmazonGlacierClient.UploadArchiveAsync\(\)](#), et ainsi de suite.
- Les versions du .NET Framework cible AWS SDK pour .NET ont défini `Timeout` et `ReadWriteTimeout` atteint les valeurs maximales pour tous les clients et objets [AmazonS3Client](#). [AmazonGlacierClient](#)
- Les versions AWS SDK pour .NET qui ciblent la bibliothèque de classes portable (PCL) et .NET Core sont définies `Timeout` sur la valeur maximale pour tous les clients et objets [AmazonS3Client](#). [AmazonGlacierClient](#)

L'exemple suivant vous montre comment définir le mode de nouvelle tentative standard, un maximum de 3 tentatives, un délai d'expiration de 10 secondes et un read/write délai d'expiration de 10 secondes (le cas échéant). [Le constructeur AmazonS3Client reçoit un objet AmazonS3Config.](#)

```
var s3Client = new AmazonS3Client(  
    new AmazonS3Config  
    {  
        Timeout = TimeSpan.FromSeconds(10),  
        // NOTE: The following property is obsolete for  
        //      versions of the AWS SDK pour .NET that target .NET Core.  
        ReadWriteTimeout = TimeSpan.FromSeconds(10),  
        RetryMode = RequestRetryMode.Standard,  
        MaxErrorRetry = 3  
    });
```

Paginateurs

Certains AWS services collectent et stockent une grande quantité de données, que vous pouvez récupérer à l'aide des appels d'API du AWS SDK pour .NET. Si la quantité de données que vous souhaitez récupérer devient trop importante pour un seul appel d'API, vous pouvez diviser les résultats en éléments plus faciles à gérer grâce à la pagination.

Pour vous permettre d'effectuer la pagination, les objets de demande et de réponse de nombreux clients de service du SDK fournissent un jeton de continuation (généralement nommé `NextToken`). Certains de ces clients de services fournissent également des paginateurs.

Les paginateurs vous permettent d'éviter la surcharge liée au jeton de continuation, qui peut impliquer des boucles, des variables d'état, plusieurs appels d'API, etc. Lorsque vous utilisez un paginateur, vous pouvez récupérer les données d'un AWS service par le biais d'une seule ligne de code, la déclaration d'une `foreach` boucle. Si plusieurs appels d'API sont nécessaires pour récupérer les données, le paginateur s'en charge pour vous.

Où puis-je trouver des paginateurs ?

Tous les services ne proposent pas de paginateurs. L'un des moyens de déterminer si un service fournit un paginateur pour une API particulière consiste à examiner la définition d'une classe de client de service dans la référence d'[AWS SDK pour .NET API](#).

Par exemple, si vous examinez la définition de la [AmazonCloudWatchLogsClient](#) classe, vous voyez une `Paginateors` propriété. Il s'agit de la propriété qui fournit un paginateur pour Amazon CloudWatch Logs.

Que m'apportent les paginateurs ?

Les paginateurs contiennent des propriétés qui vous permettent de voir les réponses complètes. Ils contiennent également généralement une ou plusieurs propriétés qui vous permettent d'accéder aux parties les plus intéressantes des réponses, que nous appellerons les résultats clés.

Par exemple, dans ce qui `AmazonCloudWatchLogsClient` a été mentionné précédemment, l'`Paginator` objet contient une `Responses` propriété contenant l'[DescribeLogGroupsResponse](#) objet complet issu de l'appel d'API. Cette `Responses` propriété contient, entre autres, un ensemble de groupes de journaux.

L'objet `Paginator` contient également un résultat clé nommé `LogGroups`. Cette propriété contient uniquement la partie de la réponse consacrée aux groupes de journaux. L'obtention de ce résultat clé vous permet de réduire et de simplifier votre code dans de nombreuses circonstances.

Pagination synchrone ou asynchrone

Les paginateurs fournissent des mécanismes de pagination synchrones et asynchrones. La pagination synchrone est disponible dans les projets .NET Framework 4.7.2 (ou version ultérieure). La pagination asynchrone est disponible dans les projets .NET Core (.NET Core 3.1, .NET 5, etc.).

Les opérations asynchrones et .NET Core étant recommandées, l'exemple suivant montre la pagination asynchrone. Les informations sur la façon d'effectuer les mêmes tâches à l'aide de la pagination synchrone et de .NET Framework 4.7.2 (ou version ultérieure) sont présentées après l'exemple dans [Considérations supplémentaires pour les paginateurs](#)

exemple

L'exemple suivant montre comment utiliser le pour AWS SDK pour .NET afficher une liste de groupes de journaux. Pour le contraste, l'exemple montre comment procéder avec et sans paginateurs. Avant d'examiner le code complet, présenté plus loin, considérez les extraits suivants.

Obtenir des groupes de CloudWatch logs sans paginateurs

```
// Loop as many times as needed to get all the log groups
var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
do
{
    Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
    var response = await cwClient.DescribeLogGroupsAsync(request);
    foreach(var logGroup in response.LogGroups)
    {
        Console.WriteLine($"{logGroup.LogGroupName}");
    }
    request.NextToken = response.NextToken;
} while(!string.IsNullOrEmpty(request.NextToken));
```

Obtenir des groupes de CloudWatch journaux à l'aide de paginateurs

```
// No need to loop to get all the log groups--the SDK does it for us behind the
scenes
```

```
var paginatorForLogGroups =  
    cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());  
await foreach(var logGroup in paginatorForLogGroups.LogGroups)  
{  
    Console.WriteLine(logGroup.LogGroupName);  
}
```

Les résultats de ces deux extraits étant exactement les mêmes, l'avantage de l'utilisation de pagination est clairement visible.

Note

Avant d'essayer de créer et d'exécuter le code complet, assurez-vous d'avoir [configuré votre environnement et votre projet](#).

Vous pourriez également avoir besoin du [Microsoft.Bcl.AsyncInterfaces](#) NuGet package car les paginationnaires asynchrones utilisent l'interface. `IAsyncEnumerable`

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.CloudWatch](#)

Éléments de programmation :

- Espace de noms [Amazon.CloudWatch](#)
Classe [AmazonCloudWatchLogsClient](#)
- Espace de noms [Amazon.CloudWatchLogs.Modèle](#)

Classe [DescribeLogGroupsRequest](#)

Classe [DescribeLogGroupsResponse](#)

Classe [LogGroup](#)

Code complet

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

namespace CWGetLogGroups
{
    class Program
    {
        // A small limit for demonstration purposes
        private const int LogGroupLimit = 3;

        //
        // Main method
        static async Task Main(string[] args)
        {
            var cwClient = new AmazonCloudWatchLogsClient();
            await DisplayLogGroupsWithoutPaginators(cwClient);
            await DisplayLogGroupsWithPaginators(cwClient);
        }

        //
        // Method to get CloudWatch log groups without paginators
        private static async Task DisplayLogGroupsWithoutPaginators(IAmazonCloudWatchLogs
cwClient)
        {
            Console.WriteLine("\nGetting list of CloudWatch log groups without using
paginators...");

            Console.WriteLine("-----");

            // Loop as many times as needed to get all the log groups
            var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
            do
            {
                Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
                DescribeLogGroupsResponse response = await
cwClient.DescribeLogGroupsAsync(request);
                foreach(LogGroup logGroup in response.LogGroups)
                {
                    Console.WriteLine($"{logGroup.LogGroupName}");
                }
            } while (response.NextToken != null);
        }
    }
}
```

```

    }
    request.NextToken = response.NextToken;
} while(!string.IsNullOrEmpty(request.NextToken));
}

//
// Method to get CloudWatch log groups by using paginators
private static async Task DisplayLogGroupsWithPaginators(IAmazonCloudWatchLogs
cwClient)
{
    Console.WriteLine("\nGetting list of CloudWatch log groups by using
paginators...");

    Console.WriteLine("-----");

    // Access the key results; i.e., the log groups
    // No need to loop to get all the log groups--the SDK does it for us behind the
scenes
    Console.WriteLine("\nFrom the key results...");
    Console.WriteLine("-----");
    IDescribeLogGroupsPaginator paginatorForLogGroups =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
    await foreach(LogGroup logGroup in paginatorForLogGroups.LogGroups)
    {
        Console.WriteLine(logGroup.LogGroupName);
    }

    // Access the full response
    // Create a new paginator, do NOT reuse the one from above
    Console.WriteLine("\nFrom the full response...");
    Console.WriteLine("-----");
    IDescribeLogGroupsPaginator paginatorForResponses =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
    await foreach(DescribeLogGroupsResponse response in
paginatorForResponses.Responses)
    {
        Console.WriteLine($"Content length: {response.ContentLength}");
        Console.WriteLine($"HTTP result: {response.HttpStatusCode}");
        Console.WriteLine($"Metadata: {response.ResponseMetadata}");
        Console.WriteLine("Log groups:");
        foreach(LogGroup logGroup in response.LogGroups)
        {
            Console.WriteLine($"  {logGroup.LogGroupName}");
        }
    }
}

```

```
}  
}  
}  
}  
}
```

Considérations supplémentaires pour les paginateurs

- Les paginateurs ne peuvent pas être utilisés plus d'une fois

Si vous avez besoin des résultats d'un AWS paginateur spécifique à plusieurs endroits de votre code, vous ne devez pas utiliser un objet de pagination plusieurs fois. Créez plutôt un nouveau paginateur chaque fois que vous en avez besoin. Ce concept est illustré dans l'exemple de code précédent de la `DisplayLogGroupsWithPaginators` méthode.

- Pagination synchrone

La pagination synchrone est disponible pour les projets .NET Framework 4.7.2 (ou version ultérieure).

Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

Pour ce faire, créez un projet .NET Framework 4.7.2 (ou version ultérieure) et copiez-y le code précédent. Il suffit ensuite de supprimer le `await` mot clé des deux appels de `foreach` pagination, comme indiqué dans l'exemple suivant.

```
/*await*/ foreach(var logGroup in paginatorForLogGroups.LogGroups)  
{  
    Console.WriteLine(logGroup.LogGroupName);  
}
```

Créez et exécutez le projet pour obtenir les mêmes résultats que ceux obtenus avec la pagination asynchrone.

Observabilité

L'observabilité est la mesure dans laquelle l'état actuel d'un système peut être déduit des données qu'il émet. Les données émises sont communément appelées télémétrie.

Ils AWS SDK pour .NET peuvent fournir deux signaux de télémétrie courants, les métriques et les traces, ainsi que la journalisation. Vous pouvez connecter un [TelemetryProvider](#) pour envoyer des données de télémétrie à un backend d'observabilité (tel que [AWS X-Ray](#) [Amazon CloudWatch](#)), puis agir en conséquence.

Par défaut, les signaux de télémétrie sont désactivés dans le SDK. Cette rubrique explique comment activer et configurer la sortie de télémétrie.

Ressources supplémentaires

Pour plus d'informations sur l'activation et l'utilisation de l'observabilité, consultez les ressources suivantes :

- [OpenTelemetry](#)
- Le billet de blog [Enhancing Observability in the AWS SDK pour .NET with OpenTelemetry](#).
- Le billet de blog [annonçant la disponibilité générale des OpenTelemetry bibliothèques AWS .NET](#).
- [Exportateurs pour OpenTelemetry](#)
- Pour des exemples d'observabilité dans le Outils AWS pour PowerShell, voir [Observabilité](#) dans le guide [PowerShell d'utilisation des outils](#).

Configurez un **TelemetryProvider**

Vous pouvez configurer un `TelemetryProvider` dans votre application globalement pour tous les clients de service ou pour des clients individuels, comme indiqué dans les exemples suivants. La [the section called “Fournisseurs de services de télémétrie”](#) section contient des informations sur les implémentations de télémétrie, notamment des informations sur les implémentations fournies avec le SDK.

Configuration du fournisseur de télémétrie global par défaut

Par défaut, chaque client de service tente d'utiliser le fournisseur de télémétrie disponible dans le monde entier. De cette façon, vous pouvez définir le fournisseur une seule fois, et tous les clients l'utiliseront. Cela ne doit être fait qu'une seule fois, avant de créer des clients de service.

L'extrait de code suivant explique comment définir le fournisseur de télémétrie global. Il crée ensuite un client de service Amazon S3 et tente d'effectuer une opération qui échoue. Le code ajoute à la fois le suivi et les métriques à l'application. Ce code utilise les NuGet packages suivants : `OpenTelemetry.Exporter.Console` et `OpenTelemetry.Instrumentation.AWS`.

Note

Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veuillez également à ajouter `AWSSDK.SSO` et `AWSSDK.SSO0IDC`.

```
using Amazon.S3;
using OpenTelemetry;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;

Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("Demo0tel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

Sdk.CreateMeterProviderBuilder()
    .ConfigureResource(e => e.AddService("Demo0tel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

var s3Client = new AmazonS3Client();

try
{
    var listBucketsResponse = await s3Client.ListBucketsAsync();
    // Attempt to delete a bucket that doesn't exist.
```

```
var deleteBucketResponse = await s3Client.DeleteBucketAsync("amzn-s3-demo-bucket");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

Console.Read();
```

Configuration d'un fournisseur de télémétrie pour un client de service spécifique

Vous pouvez configurer un client de service individuel auprès d'un fournisseur de télémétrie spécifique (autre que le fournisseur global). Pour ce faire, utilisez la `TelemetryProvider` classe de l'objet `Config` d'un constructeur de client de service. Par exemple, consultez [AmazonS3Config](#) et recherchez la propriété `TelemetryProvider`. Consultez [the section called “Fournisseurs de services de télémétrie”](#) pour plus d'informations sur les implémentations de télémétrie personnalisées.

Rubriques

- [Métriques](#)
- [Fournisseurs de services de télémétrie](#)

Métriques

Le tableau suivant répertorie les mesures de télémétrie émises par le SDK. [Configurez un fournisseur de télémétrie](#) pour rendre les métriques observables.

Quels indicateurs sont émis ?

Nom de la métrique	Unités	Type	Attributs	Description
durée de l'appel client	s	Histogramme	rpc.service, rpc.method	Durée totale de l'appel (y compris les nouvelles tentatives, le temps d'envoi ou de réception de la demande et le corps de réponse)
client.uptime	s	Histogramme	rpc. service	Le temps écoulé depuis la création d'un client

Nom de la métrique	Unités	Type	Attributs	Description
client.call.attempts	{tentative}	Monoton Counter	rpc.service, rpc.method	Le nombre de tentatives pour une opération individuelle
client.call.errors	{erreur}	Monoton Counter	rpc.service, rpc.method, exception.type	Le nombre d'erreurs associées à une opération
client.call.request_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour se connecter au service, envoyer la demande et récupérer le code d'état HTTP et les en-têtes (y compris le temps passé dans la file d'attente avant d'être envoyé)
client.call.resolve_endpoint_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour résoudre un point de terminaison (résolveur de point de terminaison, pas DNS) pour la demande
client.call.serialization_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour sérialiser le corps d'un message
client.call.deserialization_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour désérialiser le corps d'un message
client.call.auth.signing_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour signer une demande
client.call.auth.resolve_identity_duration	s	Histogramme	rpc.service, rpc.method	Le temps nécessaire pour acquérir une identité (telle qu'un AWS identifiant ou un jeton porteur) auprès d'un fournisseur d'identité
client.http.bytes_sent	Par	Monoton Counter	adresse du serveur	Le nombre total d'octets envoyés par le client HTTP

Nom de la métrique	Unités	Type	Attributs	Description
client.http.octets_received	Par	Monoton Counter	adresse du serveur	Le nombre total d'octets reçus par le client HTTP

Les descriptions des colonnes sont les suivantes :

- Nom de la métrique : nom de la métrique émise.
- Unités : unité de mesure de la métrique. Les unités sont données dans la notation [UCUM](#) sensible aux majuscules et minuscules (« c/s »).
- Type : type d'instrument utilisé pour capturer la métrique.
- Attributs : ensemble d'attributs (dimensions) émis avec la métrique.
- Description —Description de ce que mesure la métrique.

Fournisseurs de services de télémétrie

[Le SDK fournit une implémentation en OpenTelemetry tant que fournisseur de télémétrie, décrite dans la section suivante.](#)

Si vous avez des exigences spécifiques en matière de télémétrie, si vous avez déjà une solution de télémétrie en tête ou si vous avez besoin d'un contrôle précis sur la manière dont les données de télémétrie sont capturées et traitées, vous pouvez également mettre en œuvre votre propre fournisseur de télémétrie.

Enregistrez votre propre implémentation auprès de la [TelemetryProvider](#) classe. Voici un exemple simple de la façon d'enregistrer votre propre `TracerProvider` et `MeterProvider`.

```
using Amazon;
using Amazon.Runtime.Telemetry;
using Amazon.Runtime.Telemetry.Metrics;
using Amazon.Runtime.Telemetry.Tracing;

public class CustomTracerProvider : TracerProvider
{
    // Implement custom tracing logic here
}
```



```
public class CustomMeterProvider : MeterProvider
{
    // Implement custom metrics logic here
}

// Register custom implementations
AWSConfigs.TelemetryProvider.RegisterTracerProvider(new CustomTracerProvider());
AWSConfigs.TelemetryProvider.RegisterMeterProvider(new CustomMeterProvider());
```

Rubriques

- [Configuration du fournisseur OpenTelemetry de télémétrie basé](#)

Configuration du fournisseur OpenTelemetry de télémétrie basé

AWS SDK pour .NET Cela inclut la mise en œuvre d'un fournisseur de télémétrie OpenTelemetry basé. Pour plus de détails sur la façon de définir ce fournisseur en tant que fournisseur de télémétrie mondial, consultez [Configurez un TelemetryProvider](#) Pour utiliser ce fournisseur de télémétrie, vous avez besoin des ressources suivantes dans votre projet :

- Le package [OpenTelemetry.Instrumentation.aws](#). NuGet
- Un exportateur de télémétrie tel que OTLP ou Console. Pour plus d'informations, consultez la section [Exportateurs](#) dans la OpenTelemetry documentation.

L' OpenTelemetry implémentation incluse dans le SDK peut être configurée pour réduire le suivi des requêtes HTTPS, des informations d'identification et de la compression. Pour ce faire, définissez l'`SuppressDownstreamInstrumentation` sur `true`, comme suit :

```
Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation(options => options.SuppressDownstreamInstrumentation = true)
    .AddConsoleExporter()
    .Build();
```

Pour plus d'informations sur ce fournisseur, consultez le billet de blog [Enhancing Observability in the AWS SDK pour .NET with OpenTelemetry](#).

Outils supplémentaires

Voici quelques outils supplémentaires que vous pouvez utiliser pour faciliter le développement, le déploiement et la maintenance de vos applications .NET.

AWS Outil de déploiement

Après avoir développé votre application .NET Core native pour le cloud sur une machine de développement, vous pouvez utiliser l'outil de AWS déploiement pour la CLI .NET afin de déployer plus facilement votre application sur AWS.

Pour de plus amples informations, veuillez consulter [Déployez des applications pour AWS](#).

AWS Framework de traitement des messages pour .NET

Si vous utilisez des services tels qu'Amazon SQS, Amazon SNS ou EventBridge Amazon, vous pouvez peut-être tirer parti de l'infrastructure de traitement AWS des messages pour .NET. Pour de plus amples informations, veuillez consulter [AWS Framework de traitement des messages pour .NET](#).

Intégrations avec .NET Aspire

Vous pouvez tirer parti des intégrations avec .NET Aspire pour améliorer la boucle de développement interne. Pour de plus amples informations, veuillez consulter [Intégration AWS à .NET Aspire dans le AWS SDK pour .NET](#).

Authentification et autorisation avancées avec le AWS SDK pour .NET

Les rubriques de cette section fournissent des informations sur les techniques avancées d'authentification et d'autorisation dans votre AWS SDK pour .NET application.

Rubriques

- [Authentification unique avec le AWS SDK pour .NET](#)

Authentification unique avec le AWS SDK pour .NET

AWS IAM Identity Center est un service d'authentification unique (SSO) basé sur le cloud qui facilite la gestion centralisée de l'accès SSO à toutes vos applications Comptes AWS et à celles du cloud. Pour plus de détails, consultez le [guide de l'utilisateur d'IAM Identity Center](#).

Si vous ne savez pas comment un SDK interagit avec IAM Identity Center, consultez les informations suivantes.

Schéma d'interaction de haut niveau

À un niveau élevé, SDKs interagissez avec IAM Identity Center de la même manière que le modèle suivant :

1. IAM Identity Center est configuré, généralement via la [console IAM Identity Center](#), et un utilisateur SSO est invité à participer.
2. Le AWS config fichier partagé sur l'ordinateur de l'utilisateur est mis à jour avec les informations SSO.
3. L'utilisateur se connecte via IAM Identity Center et reçoit des informations d'identification à court terme pour les autorisations Gestion des identités et des accès AWS (IAM) qui ont été configurées pour lui. Cette connexion peut être initiée via un outil non SDK tel que le AWS CLI, ou par programmation via une application .NET.
4. L'utilisateur poursuit son travail. Lorsqu'ils exécutent d'autres applications utilisant l'authentification unique, ils n'ont pas besoin de se reconnecter pour ouvrir les applications.

Le reste de cette rubrique fournit des informations de référence pour la configuration et l'utilisation AWS IAM Identity Center. Il fournit des informations supplémentaires et plus avancées que la

configuration SSO de base dans [Configurer l'authentification du SDK](#). Si vous débutez dans l'utilisation de l'authentification AWS unique, vous devriez d'abord consulter cette rubrique pour obtenir des informations de base, puis suivre les didacticiels suivants pour voir l'authentification unique en action :

- [Tutoriel : application .NET uniquement](#)
- [Tutoriel : AWS CLI et application .NET](#)

Cette rubrique contient les sections suivantes :

- [Prérequis](#)
- [Configuration d'un profil SSO](#)
- [Génération et utilisation de jetons SSO](#)
- [Ressources supplémentaires](#)
- [Didacticiels](#)

Prérequis

Avant d'utiliser IAM Identity Center, vous devez effectuer certaines tâches, telles que le choix d'une source d'identité et la configuration des applications Comptes AWS et applications pertinentes. Pour plus d'informations, consultez les éléments suivants :

- Pour obtenir des informations générales sur ces tâches, consultez [Getting started](#) dans le guide de l'utilisateur d'IAM Identity Center.
- Pour des exemples de tâches spécifiques, consultez la liste des didacticiels à la fin de cette rubrique. Assurez-vous toutefois de consulter les informations contenues dans cette rubrique avant d'essayer les didacticiels.

Configuration d'un profil SSO

Une fois le centre d'identité IAM [configuré](#) dans le cas correspondant Compte AWS, un profil nommé pour l'authentification unique doit être ajouté au fichier partagé AWS `config` de l'utilisateur. Ce profil est utilisé pour se connecter au [portail AWS d'accès](#), qui renvoie des informations d'identification à court terme pour les autorisations IAM configurées pour l'utilisateur.

Le config fichier partagé est généralement nommé %USERPROFILE%\aws\config sous Windows, Linux et macOS. ~/.aws/config Vous pouvez utiliser votre éditeur de texte préféré pour ajouter un nouveau profil d'authentification unique. Vous pouvez également utiliser la `aws configure sso` commande. Pour plus d'informations sur cette commande, consultez [la section Configuration de la AWS CLI pour utiliser IAM Identity Center](#) dans le guide de l'AWS Command Line Interface utilisateur.

Le nouveau profil est similaire au suivant :

```
[profile my-sso-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-west-2
sso_account_id = 123456789012
sso_role_name = SSORedOnlyRole
```

Les paramètres du nouveau profil sont définis ci-dessous. Les deux premiers paramètres définissent le portail AWS d'accès. Les deux autres paramètres constituent une paire qui, ensemble, définit les autorisations configurées pour un utilisateur. Les quatre réglages sont obligatoires.

sso_start_url

URL pointant vers le [portail d'AWS accès de l'organisation](#). Pour trouver cette valeur, ouvrez la [console IAM Identity Center](#), sélectionnez Paramètres et recherchez l'URL du portail.

sso_region

Celui Région AWS qui contient l'hôte du portail d'accès. Il s'agit de la région qui a été sélectionnée lorsque vous avez activé IAM Identity Center. Elle peut être différente des régions que vous utilisez pour d'autres tâches.

Pour une liste complète des Régions AWS et de leurs codes, consultez la section [Points de terminaison régionaux](#) dans le Référence générale d'Amazon Web Services.

sso_account_id

L'ID d'un Compte AWS fichier ajouté via le AWS Organizations service. Pour consulter la liste des comptes disponibles, accédez à la [console IAM Identity Center](#) et ouvrez la Comptes AWS page. L'identifiant de compte que vous choisissez pour ce paramètre correspond à la valeur que vous prévoyez d'attribuer au `sso_role_name` paramètre, qui est indiquée ci-dessous.

sso_role_name

Nom d'un ensemble d'autorisations IAM Identity Center. Cet ensemble d'autorisations définit les autorisations accordées à un utilisateur via IAM Identity Center.

La procédure suivante permet de trouver la valeur de ce paramètre.

1. Accédez à la [console IAM Identity Center](#) et ouvrez la Comptes AWSpage.
2. Choisissez un compte pour afficher ses informations. Le compte que vous choisirez sera celui qui contient l'utilisateur ou le groupe SSO auquel vous souhaitez accorder des autorisations SSO.
3. Consultez la liste des utilisateurs et des groupes affectés au compte et trouvez l'utilisateur ou le groupe qui vous intéresse. L'ensemble d'autorisations que vous spécifiez dans le `sso_role_name` paramètre est l'un des ensembles associés à cet utilisateur ou à ce groupe.

Lorsque vous attribuez une valeur à ce paramètre, utilisez le nom de l'ensemble d'autorisations, et non l'Amazon Resource Name (ARN).

Les ensembles d'autorisations sont associés à des politiques IAM et à des politiques d'autorisations personnalisées. Pour plus d'informations, consultez la section [Ensembles d'autorisations](#) dans le guide de l'utilisateur d'IAM Identity Center.

Génération et utilisation de jetons SSO

Pour utiliser l'authentification unique, un utilisateur doit d'abord générer un jeton temporaire, puis utiliser ce jeton pour accéder aux AWS applications et aux ressources appropriées. Pour les applications .NET, vous pouvez utiliser les méthodes suivantes pour générer et utiliser ces jetons temporaires :

- Créez des applications .NET qui génèrent d'abord un jeton, si nécessaire, puis utilisez-le.
- Générez un jeton avec le, AWS CLI puis utilisez-le dans les applications .NET.

Ces méthodes sont décrites dans les sections suivantes et démontrées dans les [didacticiels](#).

Important

Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Application .NET uniquement

Cette section explique comment créer une application .NET qui génère un jeton SSO temporaire, si nécessaire, puis utilise ce jeton. Pour un didacticiel complet de ce processus, voir [Tutoriel pour l'authentification unique utilisant uniquement des applications .NET](#).

Générer et utiliser un jeton SSO par programmation

En plus d'utiliser le AWS CLI, vous pouvez également générer un jeton SSO par programmation.

Pour ce faire, votre application crée un [AWSCredentials](#) objet pour le profil SSO, qui charge les informations d'identification temporaires si elles sont disponibles. Votre application doit ensuite convertir l'`AWSCredentials` objet en [SSOAWSCredentials](#) objet et définir certaines propriétés d'[options](#), notamment une méthode de rappel utilisée pour demander à l'utilisateur de fournir des informations de connexion, si nécessaire.

Cette méthode est illustrée dans l'extrait de code suivant.

Important

Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

```
static AWSCredentials LoadSsoCredentials()  
{
```

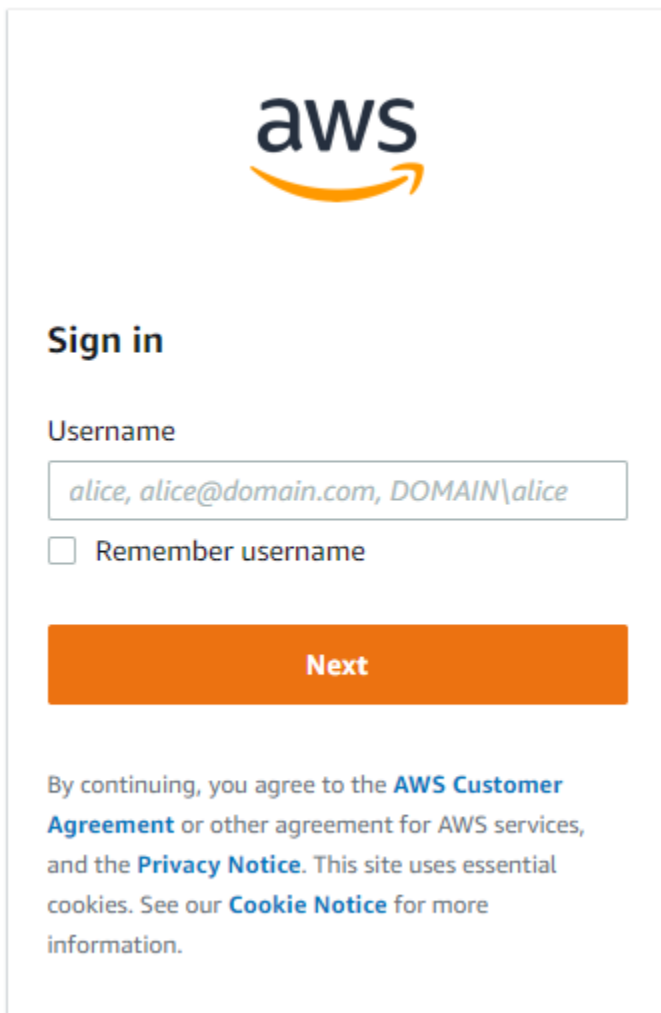
```
var chain = new CredentialProfileStoreChain();
if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
    throw new Exception("Failed to find the my-sso-profile profile");

var ssoCredentials = credentials as SSOAWSCredentials;

ssoCredentials.Options.ClientName = "Example-SSO-App";
ssoCredentials.Options.SsoVerificationCallback = args =>
{
    // Launch a browser window that prompts the SSO user to complete an SSO sign-
in.
    // This method is only invoked if the session doesn't already have a valid SSO
token.
    // NOTE: Process.Start might not support launching a browser on macOS or Linux.
If not,
    //      use an appropriate mechanism on those systems instead.
    Process.Start(new ProcessStartInfo
    {
        FileName = args.VerificationUriComplete,
        UseShellExecute = true
    });
};

return ssoCredentials;
}
```

Si aucun jeton SSO approprié n'est disponible, la fenêtre du navigateur par défaut est ouverte et la page de connexion appropriée est ouverte. Par exemple, si vous utilisez IAM Identity Center comme source d'identité, l'utilisateur voit une page de connexion similaire à la suivante :



The image shows a screenshot of the AWS sign-in page. At the top is the AWS logo. Below it is the heading "Sign in". There is a "Username" label followed by a text input field containing the placeholder text "alice, alice@domain.com, DOMAIN\alice". Below the input field is a checkbox labeled "Remember username". A large orange button with the text "Next" is positioned below the checkbox. At the bottom, there is a paragraph of text: "By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information."

Note

La chaîne de texte que vous indiquez ne `SSOAWSCredentials.Options.ClientName` peut pas comporter d'espaces. Si la chaîne contient des espaces, vous obtiendrez une exception d'exécution.

[Tutoriel pour l'authentification unique utilisant uniquement des applications .NET](#)

AWS CLI et application .NET

Cette section explique comment générer un jeton SSO temporaire à l'aide du AWS CLI, et comment utiliser ce jeton dans une application. Pour un didacticiel complet de ce processus, voir [Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET](#).

Générez un jeton SSO à l'aide du AWS CLI

En plus de générer un jeton SSO temporaire par programmation, vous utilisez le AWS CLI pour générer le jeton. Les informations suivantes vous montrent comment procéder.

Une fois que l'utilisateur a créé un profil compatible SSO, comme indiqué dans la [section précédente](#), il exécute la `aws sso login` commande à partir du AWS CLI. Ils doivent veiller à inclure le `--profile` paramètre dans le nom du profil compatible SSO, comme illustré dans l'exemple suivant :

```
aws sso login --profile my-sso-profile
```

Si l'utilisateur souhaite générer un nouveau jeton temporaire après l'expiration du jeton actuel, il peut exécuter à nouveau la même commande.

Utiliser le jeton SSO généré dans une application .NET

Les informations suivantes vous montrent comment utiliser un jeton temporaire déjà généré.

Important

Votre application doit référencer les NuGet packages suivants pour que la résolution SSO puisse fonctionner :

- `AWSSDK.SSO`
- `AWSSDK.SSO0IDC`

Le fait de ne pas référencer ces packages entraînera une exception d'exécution.

Votre application crée un [AWSCredentials](#) objet pour le profil SSO, qui charge les informations d'identification temporaires générées précédemment par le AWS CLI. Cette méthode est similaire aux méthodes présentées dans [Accès aux informations d'identification et aux profils dans une application](#) et se présente sous la forme suivante :

```
static AWSCredentials LoadSsoCredentials()  
{  
    var chain = new CredentialProfileStoreChain();  
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
```

```
        throw new Exception("Failed to find the my-sso-profile profile");  
  
    return credentials;  
}
```

L'`AWSCredentials` objet est ensuite transmis au constructeur pour un client de service. Par exemple :

```
var S3Client_SS0 = new AmazonS3Client(LoadSsoCredentials());
```

Note

L'utilisation `AWSCredentials` pour charger des informations d'identification temporaires n'est pas nécessaire si votre application a été conçue pour utiliser le [default] profil pour l'authentification unique. Dans ce cas, l'application peut créer des clients de AWS service sans paramètres, comme « `var client = new AmazonS3Client();` ».

[Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET](#)

Ressources supplémentaires

Pour obtenir de l'aide supplémentaire, consultez les ressources suivantes.

- [Qu'est-ce que IAM Identity Center ?](#)
- [Configuration du AWS CLI pour utiliser IAM Identity Center](#)
- [Utilisation des informations d'identification IAM Identity Center dans le AWS Toolkit for Visual Studio](#)

Didacticiels

Rubriques

- [Tutoriel pour l'authentification unique utilisant uniquement des applications .NET](#)
- [Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET](#)

Tutoriel pour l'authentification unique utilisant uniquement des applications .NET

Ce didacticiel explique comment activer l'authentification unique pour une application de base et pour un utilisateur d'authentification unique de test. [Il configure l'application pour générer un jeton SSO temporaire par programmation au lieu d'utiliser le. AWS CLI](#)

Ce didacticiel présente une petite partie des fonctionnalités SSO du AWS SDK pour .NET. Pour plus de détails sur l'utilisation d'IAM Identity Center avec le AWS SDK pour .NET, consultez la rubrique contenant des [informations générales](#). Dans cette rubrique, consultez en particulier la description détaillée de ce scénario dans la sous-section intitulée [Application .NET uniquement](#).

Note

Plusieurs étapes de ce didacticiel vous aident à configurer des services tels que AWS Organizations IAM Identity Center. Si vous avez déjà effectué cette configuration, ou si vous êtes uniquement intéressé par le code, vous pouvez passer à la section contenant [l'exemple de code](#).

Conditions préalables

- Configurez votre environnement de développement si ce n'est pas déjà fait. Ceci est décrit dans les sections telles que [Installez et configurez votre chaîne d'outils](#) et [Mise en route](#).
- Identifiez ou créez-en au moins un Compte AWS que vous pouvez utiliser pour tester l'authentification unique. Dans le cadre de ce didacticiel, cela s'appelle le compte de test Compte AWS ou simplement le compte de test.
- Identifiez un utilisateur SSO qui peut tester le SSO pour vous. Il s'agit d'une personne qui utilisera le SSO et les applications de base que vous créez. Pour ce didacticiel, cette personne peut être vous (le développeur) ou quelqu'un d'autre. Nous recommandons également une configuration dans laquelle l'utilisateur SSO travaille sur un ordinateur qui ne se trouve pas dans votre environnement de développement. Cependant, cela n'est pas strictement nécessaire.
- L'ordinateur de l'utilisateur SSO doit disposer d'un framework .NET compatible avec celui que vous avez utilisé pour configurer votre environnement de développement.

Configurez AWS

Cette section explique comment configurer différents AWS services pour ce didacticiel.

Pour effectuer cette configuration, connectez-vous d'abord au test en Compte AWS tant qu'administrateur. Ensuite, procédez comme suit :

Amazon S3

Accédez à la [console Amazon S3](#) et ajoutez des compartiments inoffensifs. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces buckets.

AWS IAM

Accédez à la [console IAM](#) et ajoutez quelques utilisateurs IAM. Si vous accordez des autorisations aux utilisateurs IAM, limitez les autorisations à quelques autorisations inoffensives en lecture seule. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces utilisateurs IAM.

AWS Organizations

Accédez à la [AWS Organizations console](#) et activez Organizations. Pour plus d'informations, consultez [Création d'une organisation](#) dans le [AWS Organizations Guide de l'utilisateur](#).

Cette action ajoute le test Compte AWS à l'organisation en tant que compte de gestion. Si vous avez des comptes de test supplémentaires, vous pouvez les inviter à rejoindre l'organisation, mais cela n'est pas nécessaire pour ce didacticiel.

IAM Identity Center

Accédez à la [console IAM Identity Center](#) et activez l'authentification unique. Effectuez la vérification des e-mails si nécessaire. Pour plus d'informations, consultez la section [Activer le centre d'identité IAM](#) dans le guide de l'[utilisateur d'IAM Identity Center](#).

Effectuez ensuite la configuration suivante.

Configuration du centre d'identité IAM

1. Accédez à la page des paramètres. Recherchez « l'URL du portail d'accès » et enregistrez la valeur pour une utilisation ultérieure dans le `sso_start_url` paramètre.
2. Dans le bandeau du AWS Management Console, recherchez Région AWS celui qui a été défini lorsque vous avez activé l'authentification unique. Il s'agit du menu déroulant situé à gauche de

l' Compte AWS identifiant. Enregistrez le code de région pour une utilisation ultérieure dans le `sso_region` réglage. Ce code sera similaire à `us-east-1`.

3. Créez un utilisateur SSO comme suit :
 - a. Accédez à la page Utilisateurs.
 - b. Choisissez Ajouter un utilisateur et entrez le nom d'utilisateur, l'adresse e-mail, le prénom et le nom de famille de l'utilisateur. Ensuite, choisissez Suivant.
 - c. Choisissez Suivant sur la page des groupes, puis passez en revue les informations et choisissez Ajouter un utilisateur.
4. Créez un groupe comme suit :
 - a. Accédez à la page Groupes.
 - b. Choisissez Créer un groupe et entrez le nom et la description du groupe.
 - c. Dans la section Ajouter des utilisateurs au groupe, sélectionnez l'utilisateur SSO de test que vous avez créé précédemment. Sélectionnez ensuite Créer un groupe.
5. Créez un ensemble d'autorisations comme suit :
 - a. Accédez à la page Ensembles d'autorisations et choisissez Créer un ensemble d'autorisations.
 - b. Sous Type d'ensemble d'autorisations, sélectionnez Ensemble d'autorisations personnalisé, puis cliquez sur Suivant.
 - c. Ouvrez Inline policy et entrez la politique suivante :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

- d. Pour ce didacticiel, entrez le `SSOReadOnlyRole` nom du jeu d'autorisations. Ajoutez une description si vous le souhaitez, puis choisissez Next.
 - e. Passez en revue les informations, puis choisissez Créer.
 - f. Enregistrez le nom de l'ensemble d'autorisations pour une utilisation ultérieure dans le `sso_role_name` paramètre.
6. Accédez à la page AWS des comptes et choisissez le AWS compte que vous avez ajouté à l'organisation plus tôt.
 7. Dans la section Aperçu de cette page, recherchez l'identifiant du compte et enregistrez-le pour une utilisation ultérieure dans le `sso_account_id` paramètre.
 8. Choisissez l'onglet Utilisateurs et groupes, puis choisissez Attribuer des utilisateurs ou des groupes.
 9. Sur la page Attribuer des utilisateurs et des groupes, choisissez l'onglet Groupes, sélectionnez le groupe que vous avez créé précédemment, puis cliquez sur Suivant.
 10. Sélectionnez l'ensemble d'autorisations que vous avez créé précédemment et choisissez Suivant, puis Soumettre. La configuration prend quelques instants.

Création d'exemples d'applications

Créez les applications suivantes. Ils seront exécutés sur l'ordinateur de l'utilisateur SSO.

Répertorier les compartiments Amazon S3

Incluez `AWSSDK.S3` les NuGet packages et `AWSSDK.SSO0IDC` en plus de `AWSSDK.S3` et `AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;
using System.Diagnostics;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSO0IDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
```

```

using Amazon.SecurityToken.Model;

namespace SS0Example.S3.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SS0 profile in the SS0 user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SS0 credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\\nSS0 Profile:\\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SS0 credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        // Method to get SS0 credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SS0AWSCredentials;

```



```

        ssoCredentials.Options.ClientName = "Example-SSO-App";
        ssoCredentials.Options.SsoVerificationCallback = args =>
        {
            // Launch a browser window that prompts the SSO user to complete an SSO
login.
            // This method is only invoked if the session doesn't already have a
valid SSO token.
            // NOTE: Process.Start might not support launching a browser on macOS
or Linux. If not,
            //     use an appropriate mechanism on those systems instead.
            Process.Start(new ProcessStartInfo
            {
                FileName = args.VerificationUriComplete,
                UseShellExecute = true
            });
        };

        return ssoCredentials;
    }

}

// Class to read the caller's identity.
public static class Extensions
{
    public static async Task<string> GetCallerIdentityArn(this
IAmazonSecurityTokenService stsClient)
    {
        var response = await stsClient.GetCallerIdentityAsync(new
GetCallerIdentityRequest());
        return response.Arn;
    }
}
}

```

Répertorier les utilisateurs IAM

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSO0IDC` en plus de `AWSSDK.IdentityManagement` et `AWSSDK.SecurityToken`.

```

using System;
using System.Threading.Tasks;

```

```
using System.Diagnostics;

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
// AWSSDK.SSO0IDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\\nSSO Profile:\\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }
    }
}
```

```

        // Method to get SSO credentials from the information in the shared config
        file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
                login.
                // This method is only invoked if the session doesn't already have a
                valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
                or Linux. If not,
                //      use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };

            return ssoCredentials;
        }

    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
        IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
            GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}

```

```
}
```

En plus d'afficher des listes de buckets Amazon S3 et d'utilisateurs IAM, ces applications affichent l'ARN d'identité utilisateur pour le profil compatible SSO, décrit dans ce didacticiel. `my-sso-profile`

Ces applications exécutent des tâches de connexion SSO en fournissant une méthode de rappel dans la propriété [Options d'un objet](#). [SSOAWSCredentials](#)

Instruire l'utilisateur SSO

Demandez à l'utilisateur SSO de vérifier ses e-mails et d'accepter l'invitation SSO. Ils sont invités à définir un mot de passe. Le message peut prendre quelques minutes pour arriver dans la boîte de réception de l'utilisateur SSO.

Fournissez à l'utilisateur SSO les applications que vous avez créées précédemment.

Demandez ensuite à l'utilisateur SSO d'effectuer les opérations suivantes :

1. Si le dossier contenant le AWS config fichier partagé n'existe pas, créez-le. Si le dossier existe et qu'un sous-dossier est appelé `.sso`, supprimez-le.

L'emplacement de ce dossier est généralement `%USERPROFILE%\ .aws` sous Windows, Linux et macOS. `~/ .aws`

2. Créez un AWS config fichier partagé dans ce dossier, si nécessaire, et ajoutez-y un profil comme suit :

```
[default]
region = <default Region>

[profile my-sso-profile]
sso_start_url = <user portal URL recorded earlier>
sso_region = <Region code recorded earlier>
sso_account_id = <account ID recorded earlier>
sso_role_name = SSOReadOnlyRole
```

3. Exécutez l'application Amazon S3.
4. Sur la page de connexion Web qui s'affiche, connectez-vous. Utilisez le nom d'utilisateur indiqué dans le message d'invitation et le mot de passe créé en réponse au message.
5. Lorsque la connexion est terminée, l'application affiche la liste des compartiments S3.

6. Exécutez l'application IAM. L'application affiche la liste des utilisateurs IAM. Cela est vrai même si aucune deuxième connexion n'a été effectuée. L'application IAM utilise le jeton temporaire créé précédemment.

Nettoyage

Si vous ne souhaitez pas conserver les ressources que vous avez créées au cours de ce didacticiel, nettoyez-les. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Tutoriel pour l'authentification unique à l'aide des applications .NET AWS CLI et .NET

Ce didacticiel explique comment activer l'authentification unique pour une application .NET de base et pour un utilisateur d'authentification unique de test. Il utilise le AWS CLI pour générer un jeton SSO temporaire au lieu de le [générer par programmation](#).

Ce didacticiel présente une petite partie des fonctionnalités SSO du AWS SDK pour .NET. Pour plus de détails sur l'utilisation d'IAM Identity Center avec le AWS SDK pour .NET, consultez la rubrique contenant des [informations générales](#). Dans cette rubrique, consultez en particulier la description détaillée de ce scénario dans la sous-section intitulée [AWS CLI et application .NET](#).

Note

Plusieurs étapes de ce didacticiel vous aident à configurer des services tels que AWS Organizations IAM Identity Center. Si vous avez déjà effectué ces configurations, ou si vous êtes uniquement intéressé par le code, vous pouvez passer à la section contenant l'[exemple de code](#).

Conditions préalables

- Configurez votre environnement de développement si ce n'est pas déjà fait. Ceci est décrit dans les sections telles que [Installez et configurez votre chaîne d'outils](#) et [Mise en route](#).
- Identifiez ou créez-en au moins un Compte AWS que vous pouvez utiliser pour tester l'authentification unique. Dans le cadre de ce didacticiel, cela s'appelle le compte de test Compte AWS ou simplement le compte de test.

- Identifiez un utilisateur SSO qui peut tester le SSO pour vous. Il s'agit d'une personne qui utilisera le SSO et les applications de base que vous créez. Pour ce didacticiel, cette personne peut être vous (le développeur) ou quelqu'un d'autre. Nous recommandons également une configuration dans laquelle l'utilisateur SSO travaille sur un ordinateur qui ne se trouve pas dans votre environnement de développement. Cependant, cela n'est pas strictement nécessaire.
- L'ordinateur de l'utilisateur SSO doit disposer d'un framework .NET compatible avec celui que vous avez utilisé pour configurer votre environnement de développement.
- Assurez-vous que la AWS CLI version 2 est [installée](#) sur l'ordinateur de l'utilisateur SSO. Vous pouvez vérifier cela `aws --version` en exécutant une invite de commande ou un terminal.

Configurez AWS

Cette section explique comment configurer différents AWS services pour ce didacticiel.

Pour effectuer cette configuration, connectez-vous d'abord au test en Compte AWS tant qu'administrateur. Ensuite, procédez comme suit :

Amazon S3

Accédez à la [console Amazon S3](#) et ajoutez des compartiments inoffensifs. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces buckets.

AWS IAM

Accédez à la [console IAM](#) et ajoutez quelques utilisateurs IAM. Si vous accordez des autorisations aux utilisateurs IAM, limitez les autorisations à quelques autorisations inoffensives en lecture seule. Plus loin dans ce didacticiel, l'utilisateur SSO récupérera une liste de ces utilisateurs IAM.

AWS Organizations

Accédez à la [AWS Organizations console](#) et activez Organizations. Pour plus d'informations, consultez [Création d'une organisation](#) dans le [AWS Organizations Guide de l'utilisateur](#).

Cette action ajoute le test Compte AWS à l'organisation en tant que compte de gestion. Si vous avez des comptes de test supplémentaires, vous pouvez les inviter à rejoindre l'organisation, mais cela n'est pas nécessaire pour ce didacticiel.

IAM Identity Center

Accédez à la [console IAM Identity Center](#) et activez l'authentification unique. Effectuez la vérification des e-mails si nécessaire. Pour plus d'informations, consultez la section [Activer le centre d'identité IAM](#) dans le guide de l'[utilisateur d'IAM Identity Center](#).

Effectuez ensuite la configuration suivante.

Configuration du centre d'identité IAM

1. Accédez à la page des paramètres. Recherchez « l'URL du portail d'accès » et enregistrez la valeur pour une utilisation ultérieure dans le `sso_start_url` paramètre.
2. Dans le bandeau du AWS Management Console, recherchez Région AWS celui qui a été défini lorsque vous avez activé l'authentification unique. Il s'agit du menu déroulant situé à gauche de l' Compte AWS identifiant. Enregistrez le code de région pour une utilisation ultérieure dans le `sso_region` réglage. Ce code sera similaire à `us-east-1`.
3. Créez un utilisateur SSO comme suit :
 - a. Accédez à la page Utilisateurs.
 - b. Choisissez Ajouter un utilisateur et entrez le nom d'utilisateur, l'adresse e-mail, le prénom et le nom de famille de l'utilisateur. Ensuite, choisissez Suivant.
 - c. Choisissez Suivant sur la page des groupes, puis passez en revue les informations et choisissez Ajouter un utilisateur.
4. Créez un groupe comme suit :
 - a. Accédez à la page Groupes.
 - b. Choisissez Créer un groupe et entrez le nom et la description du groupe.
 - c. Dans la section Ajouter des utilisateurs au groupe, sélectionnez l'utilisateur SSO de test que vous avez créé précédemment. Sélectionnez ensuite Créer un groupe.
5. Créez un ensemble d'autorisations comme suit :
 - a. Accédez à la page Ensembles d'autorisations et choisissez Créer un ensemble d'autorisations.
 - b. Sous Type d'ensemble d'autorisations, sélectionnez Ensemble d'autorisations personnalisé, puis cliquez sur Suivant.
 - c. Ouvrez Inline policy et entrez la politique suivante :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

- d. Pour ce didacticiel, entrez le `SSOReadOnlyRole` nom du jeu d'autorisations. Ajoutez une description si vous le souhaitez, puis choisissez Next.
 - e. Passez en revue les informations, puis choisissez Créer.
 - f. Enregistrez le nom de l'ensemble d'autorisations pour une utilisation ultérieure dans le `sso_role_name` paramètre.
6. Accédez à la page AWS des comptes et choisissez le AWS compte que vous avez ajouté à l'organisation précédemment.
 7. Dans la section Aperçu de cette page, recherchez l'identifiant du compte et enregistrez-le pour une utilisation ultérieure dans le `sso_account_id` paramètre.
 8. Choisissez l'onglet Utilisateurs et groupes, puis choisissez Attribuer des utilisateurs ou des groupes.
 9. Sur la page Attribuer des utilisateurs et des groupes, choisissez l'onglet Groupes, sélectionnez le groupe que vous avez créé précédemment, puis cliquez sur Suivant.
 10. Sélectionnez l'ensemble d'autorisations que vous avez créé précédemment et choisissez Suivant, puis Soumettre. La configuration prend quelques instants.

Création d'exemples d'applications

Créez les applications suivantes. Ils seront exécutés sur l'ordinateur de l'utilisateur SSO.

Répertorier les compartiments Amazon S3

Incluez `AWSSDK.S3` les NuGet packages et `AWSSDK.SS00IDC` en plus de `AWSSDK.S3` et `AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SS0, AWSSDK.SS00IDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SS0Example.S3.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SS0 profile in the SS0 user's shared config file.
        // - An active SS0 Token.
        //   If an active SS0 token isn't available, the SS0 user should do the
        following:
        //   In a terminal, the SS0 user must call "aws sso login --profile my-sso-
        profile".

        // Class members.
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SS0 credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\\nSS0 Profile:\\n {await
            ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SS0 credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\\nGetting a list of your buckets...");
        }
    }
}
```

```

        var listResponse = await s3Client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
        foreach (S3Bucket b in listResponse.Buckets)
        {
            Console.WriteLine(b.BucketName);
        }
        Console.WriteLine();
    }

    // Method to get SSO credentials from the information in the shared config
    file.
    static AWSCredentials LoadSsoCredentials(string profile)
    {
        var chain = new CredentialProfileStoreChain();
        if (!chain.TryGetAWSCredentials(profile, out var credentials))
            throw new Exception($"Failed to find the {profile} profile");
        return credentials;
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
        IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
            GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}

```

Répertorier les utilisateurs IAM

Incluez `AWSSDK.SSO` les NuGet packages et `AWSSDK.SSO0IDC` en plus de `AWSSDK.IdentityManagement` et `AWSSDK.SecurityToken`.

```

using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
AWSSDK.SSO0IDC

```

```
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SS0Example.IAM.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //   If an active SSO token isn't available, the SSO user should do the
        following:
        //   In a terminal, the SSO user must call "aws sso login --profile my-sso-
        profile".

        // Class members.
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\\nSSO Profile:\\n {await
            ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }
    }
}
```

```

        // Method to get SSO credentials from the information in the shared config
        file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
        IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
            GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}

```

En plus d'afficher des listes de buckets Amazon S3 et d'utilisateurs IAM, ces applications affichent l'ARN d'identité utilisateur pour le profil compatible SSO, décrit dans ce didacticiel. `my-sso-profile`

Instruire l'utilisateur SSO

Demandez à l'utilisateur SSO de vérifier ses e-mails et d'accepter l'invitation SSO. Ils sont invités à définir un mot de passe. Le message peut prendre quelques minutes pour arriver dans la boîte de réception de l'utilisateur SSO.

Fournissez à l'utilisateur SSO les applications que vous avez créées précédemment.

Demandez ensuite à l'utilisateur SSO d'effectuer les opérations suivantes :

1. Si le dossier contenant le AWS config fichier partagé n'existe pas, créez-le. Si le dossier existe et qu'un sous-dossier est appelé `.sso`, supprimez-le.

L'emplacement de ce dossier est généralement `%USERPROFILE%\ .aws` sous Windows, Linux et macOS. `~/ .aws`

2. Créez un AWS config fichier partagé dans ce dossier, si nécessaire, et ajoutez-y un profil comme suit :

```
[default]
region = <default Region>

[profile my-sso-profile]
sso_start_url = <user portal URL recorded earlier>
sso_region = <Region code recorded earlier>
sso_account_id = <account ID recorded earlier>
sso_role_name = SS0ReadOnlyRole
```

3. Exécutez l'application Amazon S3. Une exception d'exécution apparaît.
4. Exécutez la AWS CLI commande suivante :

```
aws sso login --profile my-sso-profile
```

5. Sur la page de connexion Web qui s'affiche, connectez-vous. Utilisez le nom d'utilisateur indiqué dans le message d'invitation et le mot de passe créé en réponse au message.
6. Exécutez à nouveau l'application Amazon S3. L'application affiche désormais la liste des compartiments S3.
7. Exécutez l'application IAM. L'application affiche la liste des utilisateurs IAM. Cela est vrai même si aucune deuxième connexion n'a été effectuée. L'application IAM utilise le jeton temporaire créé précédemment.

Nettoyage

Si vous ne souhaitez pas conserver les ressources que vous avez créées au cours de ce didacticiel, nettoyez-les. Il peut s'agir de AWS ressources ou de ressources de votre environnement de développement, telles que des fichiers et des dossiers.

Déployez des applications pour AWS

Après avoir développé votre application ou service .NET Core natif pour le cloud sur une machine de développement, vous devez le déployer AWS sur. Vous pouvez le faire en utilisant le AWS Management Console ou certains services tels que CloudFormation ou AWS Cloud Development Kit (AWS CDK). Vous pouvez également utiliser AWS des outils créés à des fins de déploiement. À l'aide de ces outils, vous pouvez effectuer les opérations suivantes.

Déploiement à partir de la CLI .NET

Vous pouvez utiliser les AWS outils suivants pour l'interface de ligne de commande .NET afin de déployer vos applications pour AWS :

- [AWS Outil de déploiement pour .NET CLI](#) : prend en charge les déploiements vers [AWS App Runner](#) [Amazon Elastic Container Service \(Amazon ECS\)](#) et [AWS Elastic Beanstalk](#)
- [AWS Lambda Outils pour .NET CLI](#) : prend en charge les déploiements de AWS Lambda projets.

Déployez à partir des boîtes à outils de l'IDE

Vous pouvez utiliser des AWS boîtes à outils pour déployer vos applications directement depuis l'IDE de votre choix :

- [AWS Toolkit for Visual Studio](#)

Note

La fonctionnalité « Publier sur AWS » de la boîte à outils expose les mêmes fonctionnalités que l'outil de AWS déploiement pour .NET CLI. Pour en savoir plus, consultez la section [Publier sur AWS](#) dans le guide de AWS Toolkit for Visual Studio l'utilisateur.

- [AWS Toolkit for JetBrains](#)

Consultez les [sections Utilisation d'applications AWS sans serveur](#) et [Utilisation avec AWS App Runner](#).

- [AWS Boîte à outils pour VS Code](#)

Consultez les [sections Utilisation d'applications sans serveur](#) et [Utilisation AWS App Runner](#).

- [AWS Toolkit pour Azure DevOps](#)

Cas d'utilisation

Les sections suivantes contiennent des scénarios d'utilisation pour certains types d'applications, notamment des informations sur la manière dont vous utiliseriez la CLI .NET pour déployer ces applications.

- [Applications ASP.NET Core](#)
- [Applications de console .NET](#)
- [Applications Blazor WebAssembly](#)
- [AWS Lambda projets](#)

Applications ASP.NET Core

L'[outil de AWS déploiement](#) pour la CLI .NET vous aide à déployer vos applications ASP.NET et vous guide tout au long du processus de déploiement. Il s'agit d'un outil interactif pour la CLI .NET qui permet de déployer des applications .NET avec un minimum de AWS connaissances.

L'outil de déploiement possède les fonctionnalités suivantes :

- Recommandations de calcul pour votre application : obtenez les recommandations de calcul et découvrez quel AWS calcul convient le mieux à votre application.
- Génération de Dockerfile - L'outil génère un Dockerfile si nécessaire ou utilise un Dockerfile existant.
- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.
- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

L'[AWS outil](#) de déploiement prend en charge le déploiement d'applications ASP.NET Core sur les AWS services suivants :

- Utilisation [d'Amazon ECS Service AWS Fargate](#): prend en charge les déploiements d'applications Web sur Amazon Elastic Container Service (Amazon ECS) avec une puissance de calcul gérée par AWS Fargate un moteur de calcul sans serveur.
- [AWS App Runner](#)- Prend en charge les déploiements vers un service entièrement géré qui permet aux développeurs de déployer facilement des applications Web conteneurisées à grande échelle. APIs Aucune expérience préalable en infrastructure n'est requise.
- [AWS Elastic Beanstalk](#)- Prend en charge les déploiements vers un service qui permet aux développeurs de déployer facilement des applications Web et APIs vers un environnement entièrement géré à grande échelle. Aucune expérience préalable en infrastructure n'est requise.

Pour en savoir plus, consultez la [présentation de l'outil](#). Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez [Comment installer pour](#) obtenir les instructions d'installation.

Applications de console .NET

L'[outil de AWS déploiement](#) pour la CLI .NET vous aide à déployer vos applications de console .NET sous forme de service ou de tâche planifiée sous forme d'image de conteneur sous Linux et vous guide tout au long du processus de déploiement. Si votre application ne possède pas de Dockerfile, l'outil le génère automatiquement. Dans le cas contraire, un Dockerfile existant est utilisé.

L'outil de déploiement possède les fonctionnalités suivantes :

- Recommandations de calcul pour votre application : obtenez les recommandations de calcul et découvrez quel AWS calcul convient le mieux à votre application.
- Génération de Dockerfile - L'outil génère un Dockerfile si nécessaire ou utilise un Dockerfile existant.
- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également

contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.

- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

L'[AWS outil](#) de déploiement prend en charge le déploiement d'applications .NET Console vers les AWS services suivants :

- Utilisation du [service Amazon ECS AWS Fargate](#): prend en charge les déploiements d'applications .NET en tant que service (par exemple, un processeur d'arrière-plan) sur Amazon Elastic Container Service (Amazon ECS) avec une puissance de calcul gérée AWS Fargate par un moteur de calcul sans serveur.
- Utilisation de [tâches planifiées Amazon ECS AWS Fargate](#): prend en charge les déploiements d'applications .NET en tant que tâche planifiée (par exemple, end-of-day processus) sur Amazon ECS avec une puissance de calcul gérée par un moteur de calcul AWS Fargate sans serveur.

Pour en savoir plus, consultez la [présentation de l'outil](#). Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez [Comment installer pour](#) obtenir les instructions d'installation.

Applications Blazor WebAssembly

L'[outil de AWS déploiement](#) pour la CLI .NET vous aide à héberger votre WebAssembly application Blazor dans Amazon S3, en utilisant Amazon CloudFront pour la diffusion de contenu sur le réseau. Votre application est déployée dans un compartiment S3 pour l'hébergement Web. L'outil crée et configure un compartiment S3, puis télécharge votre application Blazor dans le compartiment.

L'outil de déploiement possède les fonctionnalités suivantes :

- Packaging et déploiement automatiques : l'outil crée les artefacts de déploiement, provisionne l'infrastructure à l'aide d'un projet de AWS CDK déploiement généré et déploie votre application sur le système de AWS calcul choisi.
- Déploiements reproductibles et partageables : vous pouvez générer et modifier des projets de AWS CDK déploiement en fonction de votre cas d'utilisation spécifique. Vous pouvez également contrôler les versions de vos projets et les partager avec votre équipe pour des déploiements reproductibles.

- Aide à l'apprentissage AWS CDK pour .NET - L'outil vous aide à apprendre progressivement les AWS outils sous-jacents sur lesquels il repose, tels que AWS CDK.

Pour en savoir plus, consultez la [présentation de l'outil](#). Pour commencer à partir de là, accédez à Documentation, Getting started, puis choisissez [Comment installer pour](#) obtenir les instructions d'installation.

AWS Lambda projets

AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Il exécute votre code sur une infrastructure informatique à haute disponibilité et exécute l'ensemble de l'administration des ressources informatiques. Pour plus d'informations sur Lambda, consultez [Qu'est-ce que AWS Lambda](#) ? dans le Guide AWS Lambda du développeur.

Vous pouvez déployer des fonctions Lambda à l'aide de l'interface de ligne de commande (CLI) .NET.

Rubriques

- [Conditions préalables](#)
- [Commandes Lambda disponibles](#)
- [Étapes de déploiement](#)

Conditions préalables

Avant de commencer à utiliser la CLI .NET pour déployer des fonctions Lambda, vous devez remplir les conditions préalables suivantes :

- Vérifiez que la CLI .NET est installée. Par exemple : `dotnet --version`. Si nécessaire, allez dans <https://dotnet.microsoft.com/download> pour l'installer.
- Configurez la CLI .NET pour qu'elle fonctionne avec Lambda. Pour une description de la procédure à suivre, consultez la [CLI .NET Core](#) dans le guide du AWS Lambda développeur. Dans cette procédure, la commande de déploiement est la suivante :

```
dotnet lambda deploy-function MyFunction --function-role role
```

Si vous ne savez pas comment créer un rôle IAM pour cet exercice, n'incluez pas cette `--function-role role` partie. L'outil vous aidera à créer un nouveau rôle.

Commandes Lambda disponibles

Pour répertorier les commandes Lambda disponibles via la CLI .NET, ouvrez une invite de commande ou un terminal et entrez `dotnet lambda --help`. Le résultat de la commande sera similaire à ce qui suit :

```
Amazon Lambda Tools for .NET applications
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Commands to deploy and manage AWS Lambda functions:

    deploy-function      Command to deploy the project to AWS Lambda
    ...
    (etc.)

To get help on individual commands execute:
    dotnet lambda help <command>
```

Le résultat répertorie toutes les commandes actuellement disponibles.

Étapes de déploiement

Les instructions suivantes supposent que vous avez créé un projet AWS Lambda .NET. Aux fins de cette procédure, le projet est nommé `DotNetCoreLambdaTest`.

1. Ouvrez une invite de commande ou un terminal, puis accédez au dossier contenant votre fichier de projet .NET Lambda.
2. Saisissez `dotnet lambda deploy-function`.
3. Si vous y êtes invité, entrez la AWS région (la région dans laquelle votre fonction Lambda sera déployée).
4. Lorsque vous y êtes invité, entrez le nom de la fonction à déployer, par exemple `DotNetCoreLambdaTest`. Il peut s'agir du nom d'une fonction qui existe déjà dans votre ordinateur Compte AWS ou d'une fonction qui n'y a pas encore été déployée.
5. Lorsque vous y êtes invité, sélectionnez ou créez le rôle IAM que Lambda assumera lors de l'exécution de la fonction.

Une fois l'opération terminée, le message Nouvelle fonction Lambda créée s'affiche.

```
Executing publish command
...
(etc.)
New Lambda function created
```

Si vous déployez une fonction qui existe déjà dans votre compte, la fonction de déploiement demande uniquement la AWS région (si nécessaire). Dans ce cas, la sortie de commande se termine par `Updating code for existing function`.

Une fois votre fonction Lambda déployée, elle est prête à être utilisée. Pour plus d'informations, consultez [Exemples d'utilisation de AWS Lambda](#).

Lambda surveille automatiquement les fonctions Lambda pour vous et fournit des statistiques via Amazon CloudWatch. Pour surveiller et résoudre les problèmes liés à votre fonction Lambda, [consultez la section Surveillance et résolution des problèmes liés aux applications Lambda](#).

Migrez votre projet pour AWS SDK pour .NET

Cette section fournit des informations sur les tâches de migration susceptibles de s'appliquer à vous, ainsi que des instructions sur la manière d'effectuer ces tâches.

Rubriques

- [Migration vers la version 3 du AWS SDK pour .NET](#)
- [Migration vers la version 3.5 du AWS SDK pour .NET](#)
- [Migration vers la version 3.7 du AWS SDK pour .NET](#)
- [Migration à partir de .NET Standard 1.3](#)

Migration vers la version 3 du AWS SDK pour .NET

Cette rubrique décrit les modifications apportées à la version 3 du SDK AWS SDK pour .NET et explique comment migrer votre code vers cette version du SDK.

À propos des AWS SDK pour .NET versions

Le AWS SDK pour .NET, initialement publié en novembre 2009, a été conçu pour .NET Framework 2.0. Depuis cette version, .NET s'est amélioré avec .NET Framework 4.0 et .NET Framework 4.5, ciblant de nouvelles plateformes : WinRT et Windows Phone.

AWS SDK pour .NET la version 2 a été mise à jour pour tirer parti des nouvelles fonctionnalités de la plateforme .NET et pour cibler WinRT et Windows Phone.

AWS SDK pour .NET la version 3 a été mise à jour pour rendre les assemblages modulaires.

Refonte de l'architecture du kit SDK

L'ensemble de la version 3 du AWS SDK pour .NET est repensé pour être modulaire. Chaque service est désormais implémenté dans son propre assembly, et non dans un seul et même assembly global. Il n'est plus nécessaire d'ajouter l'intégralité AWS SDK pour .NET à votre candidature. Vous pouvez désormais ajouter des assemblages uniquement pour les AWS services utilisés par votre application.

Évolutions

Les sections suivantes décrivent les modifications apportées à la version 3 du AWS SDK pour .NET.

AWSClientSupprimé en usine

La classe `Amazon.AWSClientFactory` a été supprimée. Désormais, pour créer le client d'un service, utilisez le constructeur correspondant. Par exemple, pour créer un `AmazonEC2Client` :

```
var ec2Client = new Amazon.EC2.AmazonEC2Client();
```

Amazon.Runtime.AssumeRoleAWSCredentials Supprimé

La `Amazon.Runtime.AssumeRoleAWSCredentials` classe a été supprimée car elle se trouvait dans un espace de noms principal mais dépendait du AWS Security Token Service, et parce qu'elle est obsolète dans le SDK depuis un certain temps. Utilisez plutôt la classe `Amazon.SecurityToken.AssumeRoleAWSCredentials`.

Suppression de la méthode SetACL de S3Link

La `S3Link` classe fait partie du `Amazon.DynamoDBv2` package et est utilisée pour stocker des objets dans Amazon S3 qui sont des références dans un élément DynamoDB. Cette fonctionnalité est utile, mais nous ne voulions pas créer de dépendance de compilation sur le `Amazon.S3` package pour DynamoDB. C'est pourquoi nous avons simplifié les méthodes `Amazon.S3` exposées de la classe `S3Link`, remplaçant la méthode `SetACL` par la méthode `MakeS3ObjectPublic`. Pour exercer un plus grand contrôle sur la liste de contrôle d'accès (ACL) de l'objet, utilisez directement le package `Amazon.S3`.

Suppression de classes de résultat obsolètes

Pour la plupart des services du AWS SDK pour .NET, les opérations renvoient un objet de réponse contenant les métadonnées de l'opération, telles que l'ID de demande et un objet de résultat. La présence à la fois d'une classe de réponse et d'une classe de résultat était redondante et exigeait plus de saisies des développeurs. Dans la version 2 du AWS SDK pour .NET, nous avons placé toutes les informations de la classe de résultat dans la classe de réponse. De même, nous avons marqué les classes de résultat comme étant obsolètes pour dissuader leur utilisation. Dans la version 3 du AWS SDK pour .NET, nous avons supprimé ces classes de résultats obsolètes afin de réduire la taille du SDK.

AWS Modifications apportées à la section Config

Il est possible d'effectuer une configuration avancée du fichier AWS SDK pour .NET via le `Web.config` fichier `App.config` or. Cette opération s'effectue via une section de configuration `<aws>` telle que celle illustrée ci-dessous, qui fait référence au nom d'assembly du SDK.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Dans la version 3 du AWS SDK pour .NET, l'AWSSDKAssemblage n'existe plus. Nous avons intégré le code commun dans l'assembly `AWSSDK.Core`. De ce fait, vous devrez remplacer les références à l'assembly `AWSSDK` dans votre fichier `App.config` ou `Web.config` par des références à l'assembly `AWSSDK.Core`, comme suit.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Vous pouvez également manipuler les paramètres de configuration à l'aide de la classe `Amazon.AWSConfigs`. Dans la version 3 du AWS SDK pour .NET, nous avons déplacé les paramètres de configuration de `DynamoDB` de `Amazon.AWSConfigs` la classe vers la classe `Amazon.AWSConfigsDynamoDB`.

Migration vers la version 3.5 du AWS SDK pour .NET

[La version 3.5 normalise AWS SDK pour .NET davantage l'expérience .NET en transférant la prise en charge de toutes les variantes du SDK hors Framework vers .NET Standard 2.0.](#) Selon votre environnement et votre base de code, pour tirer parti des fonctionnalités de la version 3.5, vous devrez peut-être effectuer certaines tâches de migration.

Cette rubrique décrit les changements qu'apporte la version 3.5 et le travail que vous pourriez devoir effectuer pour migrer votre environnement ou votre code à partir de la version 3.

Ce qui a changé pour la version 3.5

Ce qui suit décrit ce qui a changé ou n'a pas changé dans la AWS SDK pour .NET version 3.5.

.NET Framework et .NET Core

La prise en charge de .NET Framework et .NET Core n'a pas changé.

Xamarin

Les projets Xamarin (nouveaux et existants) doivent cibler .NET Standard 2.0. Consulter [Prise en charge de .NET Standard 2.0 dans Xamarin.Forms](#) et [Prise en charge de l'implémentation .NET](#).

Unity

Les applications Unity doivent cibler les profils .NET Standard 2.0 ou .NET 4.x à l'aide d'Unity 2018.1 ou ultérieure. Pour plus d'informations, consultez [Prise en charge des profils .NET](#). En outre, si vous utilisez IL2CPP pour compiler, vous devez désactiver le découpage de code en ajoutant un fichier link.xml, comme décrit dans [Référencement de la AWS SDK pour .NET norme 2.0 depuis Unity, Xamarin](#) ou UWP. Après le portage de votre code vers l'une des bases de code recommandées, votre application Unity peut accéder à tous les services offerts par le kit SDK.

Unity prenant en charge le .NET Standard 2.0, le package AWSSDK.Core du SDK version 3.5 ne contient plus de code spécifique à Unity, y compris certaines fonctionnalités de niveau supérieur. Pour une meilleure transition, l'ensemble du code Unity existant est disponible à titre de référence dans le aws-sdk-unity-net GitHub référentiel [aws/](#). Si vous trouvez des fonctionnalités manquantes qui ont un impact sur votre utilisation AWS de Unity, vous pouvez déposer une demande de fonctionnalité sur <https://github.com/aws/dotnet/issues>.

Voir aussi [Considérations spéciales relatives au support Unity](#).

Universal Windows Platform (UWP)

Ciblez votre application UWP vers [la version 16299 ou ultérieure](#) (mise à jour Fall Creators, version 1709, publiée en octobre 2017).

Windows Phone et Silverlight

La version 3.5 de AWS SDK pour .NET ne prend pas en charge ces plateformes car Microsoft ne les développe plus activement. Pour plus d'informations, consultez les ressources suivantes :

- [Fin de la prise en charge de Windows 10 Mobile](#)
- [Fin de la prise en charge de Silverlight](#)

Bibliothèques de classes portables héritées (basées sur les profils PCLs)

Envisagez de cibler votre bibliothèque vers .NET Standard. Pour plus d'informations, consultez [Comparaison avec les bibliothèques de classes portables](#) de Microsoft.

Gestionnaire de synchronisation Amazon Cognito et Amazon Mobile Analytics Manager

Les abstractions de haut niveau qui facilitent l'utilisation d'Amazon Cognito Sync et d'Amazon Mobile Analytics sont supprimées de la version 3.5 du AWS SDK pour .NET. AWS AppSync est le remplacement préféré d'Amazon Cognito Sync. Amazon Pinpoint est le produit de remplacement préféré d'Amazon Mobile Analytics.

[Si votre code est affecté par l'absence de code de bibliothèque de niveau supérieur pour Amazon Pinpoint, vous pouvez indiquer que vous vous intéressez à l'un des problèmes GitHub suivants ou aux deux <https://github.com/aws/dotnet/issues/20> AWS AppSync et \[dotnet/issues/19\]\(https://github.com/aws/dotnet/issues/19\). <https://github.com/aws/amazon-cognito-sync-manager> et Amazon Mobile Analytics Manager à partir des GitHub référentiels suivants : \[aws/amazon-cognito-sync-manager -net\]\(#\) et \[aws/-net.aws-mobile-analytics-manager\]\(#\)](#)

Migration de code synchrone

La version 3.5 AWS SDK pour .NET prend en charge à la fois .NET Framework et .NET Standard (via les versions .NET Core telles que .NET Core 3.1, .NET 5, etc.). Les variantes du SDK conformes à .NET Standard fournissent uniquement des méthodes asynchrones. Par conséquent, si vous souhaitez tirer parti de .NET Standard, vous devez modifier le code synchrone afin qu'il s'exécute de manière asynchrone.

Les extraits de code suivants montrent comment vous pouvez transformer le code synchrone en code asynchrone. Le code contenu dans ces extraits est utilisé pour afficher le nombre de compartiments Amazon S3.

Le code d'origine appelle [ListBuckets](#).

```
private static ListBucketsResponse MyListBuckets()
```

```
{
    var s3Client = new AmazonS3Client();
    var response = s3Client.ListBuckets();
    return response;
}

// From the calling function
ListBucketsResponse response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
```

Pour utiliser la version 3.5 du SDK, appelez [ListBucketsAsync](#) plutôt.

```
private static async Task<ListBucketsResponse> MyListBuckets()
{
    var s3Client = new AmazonS3Client();
    var response = await s3Client.ListBucketsAsync();
    return response;
}

// From an asynchronous calling function
ListBucketsResponse response = await MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");

// OR From a synchronous calling function
Task<ListBucketsResponse> response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
```

Migration vers la version 3.7 du AWS SDK pour .NET

À partir de la version 3.7, le .NET Standard 1.3 AWS SDK pour .NET n'est plus pris en charge.

Pour plus d'informations sur la migration depuis .NET Standard 1.3, consultez [Migration à partir de .NET Standard 1.3](#).

Migration à partir de .NET Standard 1.3

Le 27 juin 2019, Microsoft [a mis fin à la prise en charge](#) des versions .NET Core 1.0 et .NET Core 1.1. Suite à cette annonce, le support de .NET Standard 1.3 a AWS pris fin AWS SDK pour .NET le 31 décembre 2020.

AWS a continué à fournir des mises à jour de service et des correctifs de sécurité sur le .NET Standard 1.3 AWS SDK pour .NET cible jusqu'au 1er octobre 2020. Après cette date, la cible .NET Standard 1.3 est passée en mode maintenance, ce qui signifie qu'aucune nouvelle mise à jour n'a été publiée ; seuls des correctifs de bogues critiques et des correctifs de sécurité ont été AWS appliqués.

Le 31 décembre 2020, le support de .NET Standard 1.3 AWS SDK pour .NET a pris fin. Après cette date, aucun correctif de bogue ou correctif de sécurité n'a été appliqué. Les artefacts créés avec cette cible restent disponibles en téléchargement sur NuGet.

Ce que vous devez faire

- Si vous exécutez des applications qui utilisent .NET Framework, vous n'êtes pas concerné.
- Si vous exécutez des applications qui utilisent .NET Core 2.0 ou version ultérieure, vous n'êtes pas concerné.
- Si vous exécutez des applications qui utilisent .NET Core 1.0 ou .NET Core 1.1, migrez vos applications vers une version plus récente de .NET Core en suivant les [instructions de migration Microsoft](#). Nous recommandons au minimum .NET Core 3.1.
- Si vous exécutez des applications métier stratégiques qui ne peuvent pas être mises à niveau pour le moment, vous pouvez continuer à utiliser votre version actuelle du kit AWS SDK pour .NET.

Si vous avez des questions ou des inquiétudes, [contactez le support AWS](#).

Travaillez avec AWS les services du AWS SDK pour .NET

Les sections suivantes contiennent des exemples, des didacticiels, des tâches et des guides qui vous montrent comment utiliser les AWS SDK pour .NET AWS services. Ces exemples et didacticiels s'appuient sur une API AWS SDK pour .NET fournie. Pour connaître les classes et les méthodes disponibles dans l'API, consultez la [référence de l'AWS SDK pour .NET API](#).

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le [Faites une visite rapide](#) sujet. Il vous présente le SDK.

Vous pouvez trouver d'autres exemples de code dans le référentiel d'[exemples de AWS code et dans le référentiel awslabs](#) sur. GitHub

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

- [Exemples de code avec conseils pour AWS SDK pour .NET](#)
- [Utilisation AWS Lambda pour le service de calcul](#)
- [Bibliothèques et frameworks de haut niveau pour AWS SDK pour .NET](#)
- [Support pour d'autres AWS services et configurations](#)

Exemples de code avec conseils pour AWS SDK pour .NET

Les sections suivantes contiennent des exemples de code et fournissent des conseils pour les exemples. Ils peuvent vous aider à apprendre à utiliser le AWS SDK pour .NET pour utiliser les AWS services.

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le [Faites une visite rapide](#) sujet. Il vous présente le SDK.

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

- [Accès à CloudFormation l'aide du AWS SDK pour .NET](#)

- [Authentification des utilisateurs avec Amazon Cognito](#)
- [Utilisation des bases de données NoSQL Amazon DynamoDB](#)
- [Utilisation avec Amazon EC2](#)
- [Accès Gestion des identités et des accès AWS \(IAM\) à l'aide du AWS SDK pour .NET](#)
- [Utilisation du stockage sur Internet d'Amazon Simple Storage Service](#)
- [Envoyer des notifications depuis le cloud à l'aide d'Amazon Simple Notification Service](#)
- [Messagerie à l'aide d'Amazon SQS](#)

Accès à CloudFormation l'aide du AWS SDK pour .NET

Les AWS SDK pour .NET supports [AWS CloudFormation](#), qui créent et provisionnent les déploiements AWS d'infrastructures de manière prévisible et répétée.

APIs

Les AWS SDK pour .NET offres APIs pour les CloudFormation clients. Ils vous APIs permettent de travailler avec des CloudFormation fonctionnalités telles que les modèles et les piles. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la [référence de l'AWS SDK pour .NET API](#) (et faites défiler la page jusqu'à « Amazon »). CloudFormation«).

Ils AWS CloudFormation APIs sont fournis par le [AWSSDK. CloudFormation](#)colis.

Conditions préalables

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

Rubriques

- [Lister AWS les ressources à l'aide AWS CloudFormation](#)

Lister AWS les ressources à l'aide AWS CloudFormation

Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour répertorier les ressources par CloudFormation piles. L'exemple utilise l'API de bas niveau. L'application ne prend aucun

argument, mais rassemble simplement des informations pour toutes les piles accessibles aux informations d'identification de l'utilisateur, puis affiche des informations sur ces piles.

Références du SDK

NuGet colis :

- [AWSSDK.CloudFormation](#)

Éléments de programmation :

- Espace de noms [Amazon. CloudFormation](#)

Classe [AmazonCloudFormationClient](#)

- Espace de noms [Amazon. CloudFormation.Modèle](#)

Classe [ICloudFormationPaginatorFactory. DescribeStacks](#)

Classe [DescribeStackResourcesRequest](#)

Classe [DescribeStackResourcesResponse](#)

Class [Stack](#)

Classe [StackResource](#)

[Tag](#) de classe

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
    }
}
```

```

        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\\nIn Region:
{_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

                Console.WriteLine("\\n-----");
                Console.WriteLine($"\\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {

```

```

        StackName = stack.StackName
    });
    if (responseDescribeResources.StackResources.Count > 0)
    {
        Console.WriteLine("  Resources:");
        foreach (StackResource resource in responseDescribeResources
            .StackResources)
        {
            Console.WriteLine(
                $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
        }
    }

    Console.WriteLine("\n-----");
    return true;
}
catch (AmazonCloudFormationException ex)
{
    Console.WriteLine("Unable to get stack information:\n" + ex.Message);
    return false;
}
catch (AmazonServiceException ex)
{
    if (ex.Message.Contains("Unable to get IAM security credentials"))
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine("If you are usnig SS0, be sure to install" +
            " the AWSSDK.SS0 and AWSSDK.SS00IDC packages.");
    }
    else
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.StackTrace);
    }

    return false;
}
catch (ArgumentNullException ex)
{
    if (ex.Message.Contains("Options property cannot be empty: ClientName"))
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine("If you are using SS0, have you logged in?");
    }
}

```



```
        else
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine(ex.StackTrace);
        }

        return false;
    }
}
```

Authentification des utilisateurs avec Amazon Cognito

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

À l'aide d'Amazon Cognito Identity, vous pouvez créer des identités uniques pour vos utilisateurs et les authentifier pour un accès sécurisé à vos AWS ressources telles qu'Amazon S3 ou Amazon DynamoDB. Amazon Cognito Identity prend en charge les fournisseurs d'identité publics tels qu'Amazon, Facebook, Twitter/Digits, Google ou tout autre fournisseur compatible avec OpenID Connect, ainsi que les identités non authentifiées. Amazon Cognito prend également en charge les [identités authentifiées par les développeurs](#), qui vous permettent d'enregistrer et d'authentifier les utilisateurs à l'aide de votre propre processus d'authentification principal, tout en utilisant Amazon Cognito Sync pour synchroniser les données utilisateur et accéder aux ressources. AWS

Pour plus d'informations sur [Amazon Cognito](#), consultez le guide du développeur [Amazon Cognito](#).

Les exemples de code suivants montrent comment utiliser facilement Amazon Cognito Identity. L'[Fournisseur d'informations d'identification](#) exemple montre comment créer et authentifier les identités des utilisateurs. L'[CognitoAuthentication bibliothèque d'extensions](#) exemple montre comment utiliser la bibliothèque d' CognitoAuthentication extensions pour authentifier les groupes d'utilisateurs Amazon Cognito.

Rubriques

- [Fournisseur d'informations d'identification Amazon Cognito](#)
- [Exemples de bibliothèques d' CognitoAuthentication extensions Amazon](#)

Fournisseur d'informations d'identification Amazon Cognito

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, que l'on trouve dans le [AWSSDK.CognitoIdentity](#) NuGetpackage, est un objet d'identification qui utilise Amazon Cognito et the AWS Security Token Service (AWS STS) pour récupérer les informations d'identification pour passer AWS des appels.

La première étape de configuration de `CognitoAWSCredentials` consiste à créer un « groupe d'identités ». Un groupe d'identités est un magasin d'informations relatives à l'identité de l'utilisateur qui est propre au compte. Les informations peuvent être récupérées quels que soient les plateformes client, appareils et systèmes d'exploitation. Autrement dit, si un utilisateur commence à utiliser votre application sur un téléphone, puis passe à une tablette, les informations de l'application restent disponibles pour cet utilisateur. Vous pouvez créer un nouveau pool d'identités depuis la console Amazon Cognito. Si vous utilisez la console, celle-ci vous fournira également les autres éléments d'information dont vous avez besoin :

- Votre numéro de compte – Un numéro à 12 chiffres, comme 123456789012, qui est unique pour votre compte.
- L'ARN du rôle non authentifié – Un rôle que les utilisateurs non authentifiés assumeront. Par exemple, ce rôle peut fournir des autorisations d'accès en lecture seule à vos données.
- L'ARN du rôle authentifié – Un rôle que les utilisateurs authentifiés assumeront. Ce rôle peut fournir des autorisations plus étendues à vos données.

Configurer Cognito AWSCredentials

L'exemple de code suivant montre comment procéder à la configuration `CognitoAWSCredentials`, que vous pouvez ensuite utiliser pour appeler Amazon S3 en tant qu'utilisateur non authentifié. Cela vous permet d'effectuer des appels avec seulement un volume minimum de données requises pour authentifier l'utilisateur. Les autorisations utilisateur sont contrôlées par le rôle. Vous pouvez donc configurer l'accès selon vos besoins.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
```

```
    accountId,          // Account number
    identityPoolId,     // Identity pool ID
    unAuthRoleArn,      // Role for unauthenticated users
    null,               // Role for authenticated users, not set
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    s3Client.ListBuckets();
}
```

Utiliser AWS en tant qu'utilisateur non authentifié

L'exemple de code suivant montre comment vous pouvez commencer à utiliser en AWS tant qu'utilisateur non authentifié, puis vous authentifier via Facebook et mettre à jour les informations d'identification pour utiliser les informations d'identification Facebook. En utilisant cette approche, vous pouvez accorder différentes capacités aux utilisateurs authentifiés via le rôle authentifié. Par exemple, votre application de téléphone peut autoriser des utilisateurs à afficher un contenu de manière anonyme, mais leur permet de publier s'ils sont connectés avec un ou plusieurs des fournisseurs configurés.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId, identityPoolId,
    unAuthRoleArn,    // Role for unauthenticated users
    authRoleArn,      // Role for authenticated users
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    // Initial use will be unauthenticated
    s3Client.ListBuckets();

    // Authenticate user through Facebook
    string facebookToken = GetFacebookAuthToken();

    // Add Facebook login to credentials. This clears the current AWS credentials
    // and retrieves new AWS credentials using the authenticated role.
    credentials.AddLogin("graph.facebook.com", facebookAccessToken);

    // This call is performed with the authenticated role and credentials
    s3Client.ListBuckets();
}
```

L'objet `CognitoAWSCredentials` fournit encore plus de fonctionnalités si vous l'utilisez avec l'objet `AmazonCognitoSyncClient` qui fait partie du kit AWS SDK pour .NET. Si vous utilisez les deux `AmazonCognitoSyncClient` et `CognitoAWSCredentials`, vous n'avez pas à spécifier les `IdentityId` propriétés `IdentityPoolId` et lorsque vous passez des appels avec le `AmazonCognitoSyncClient`. Ces propriétés sont automatiquement renseignées à partir de `CognitoAWSCredentials`. L'exemple de code suivant illustre ceci, ainsi qu'un événement qui vous informe chaque fois que `IdentityId` pour `CognitoAWSCredentials` est modifié. `IdentityId` peut changer dans certains cas, par exemple lors du passage d'un utilisateur non authentifié à un utilisateur authentifié.

```
CognitoAWSCredentials credentials = GetCognitoAWSCredentials();

// Log identity changes
credentials.IdentityChangedEvent += (sender, args) =>
{
    Console.WriteLine("Identity changed: [{0}] => [{1}]", args.OldIdentityId,
        args.NewIdentityId);
};

using (var syncClient = new AmazonCognitoSyncClient(credentials))
{
    var result = syncClient.ListRecords(new ListRecordsRequest
    {
        DatasetName = datasetName
        // No need to specify these properties
        //IdentityId = "...",
        //IdentityPoolId = "..."
    });
}
```

Exemples de bibliothèques d' `CognitoAuthentication` extensions Amazon

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

La bibliothèque d' `CognitoAuthentication` extensions, qui se trouve dans le dossier [Amazon.Extensions.CognitoAuthentication](#) NuGet package, simplifie le processus d'authentification

des groupes d'utilisateurs Amazon Cognito pour les développeurs .NET Core et Xamarin. La bibliothèque repose sur l'API du fournisseur d'identité Amazon Cognito pour créer et envoyer des appels d'API d'authentification utilisateur.

Utilisation de la bibliothèque d' `CognitoAuthentication` extensions

Amazon Cognito intègre des `ChallengeName` valeurs `AuthFlow` et des valeurs pour un flux d'authentification standard permettant de valider le nom d'utilisateur et le mot de passe via le `Secure Remote Password (SRP)`. Pour plus d'informations sur le flux d'authentification, consultez [Flux d'authentification du groupe d'utilisateurs Amazon Cognito](#).

Les exemples suivant nécessitent ces instructions using :

```
// Required for all examples
using System;
using Amazon;
using Amazon.CognitoIdentity;
using Amazon.CognitoIdentityProvider;
using Amazon.Extensions.CognitoAuthentication;
using Amazon.Runtime;
// Required for the GetS3BucketsAsync example
using Amazon.S3;
using Amazon.S3.Model;
```

Utiliser l'authentification de base

Créez un [AmazonCognitoIdentityProviderClient](#) en utilisant [Anonymous AWSCredentials](#), qui ne nécessite pas de demandes signées. Vous n'avez pas besoin de fournir une région ; le code sous-jacent appelle `FallbackRegionFactory.GetRegionEndpoint()` si aucune région n'est indiquée. Créez des objets `CognitoUserPool` et `CognitoUser`. Appelez la méthode `StartWithSrpAuthAsync` avec une demande `InitiateSrpAuthRequest` qui contient le mot de passe utilisateur.

```
public static async void GetCredsAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
        Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest()
    {
```

```
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);
    accessToken = authResponse.AuthenticationResult.AccessToken;
}
```

Authentifiez-vous en relevant les défis

Il est également plus simple de poursuivre le flux d'authentification avec `NewPasswordRequired` des défis tels que l'authentification multifactorielle (MFA). Les seules exigences sont les `CognitoAuthentication` objets, le mot de passe de l'utilisateur pour le SRP et les informations nécessaires pour le prochain défi, qui sont acquises après avoir invité l'utilisateur à les saisir. Le code suivant montre un moyen de vérifier le type de défi et d'obtenir les réponses appropriées pour le MFA et les `NewPasswordRequired` défis pendant le flux d'authentification.

Effectuez une authentification de base comme auparavant, et attendez (`await`) une réponse `AuthFlowResponse`. Lorsque la réponse est reçue, parcourez l'objet `AuthenticationResult` renvoyé. Si le type `ChallengeName` est `NEW_PASSWORD_REQUIRED`, appelez la méthode `RespondToNewPasswordRequiredAsync`.

```
public static async void GetCredsChallengesAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest(){
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);

    while (authResponse.AuthenticationResult == null)
    {
        if (authResponse.ChallengeName == ChallengeNameType.NEW_PASSWORD_REQUIRED)
        {
            Console.WriteLine("Enter your desired new password:");
        }
    }
}
```

```
        string newPassword = Console.ReadLine();

        authResponse = await user.RespondToNewPasswordRequiredAsync(new
RespondToNewPasswordRequiredRequest()
        {
            SessionID = authResponse.SessionID,
            NewPassword = newPassword
        });
        accessToken = authResponse.AuthenticationResult.AccessToken;
    }
    else if (authResponse.ChallengeName == ChallengeNameType.SMS_MFA)
    {
        Console.WriteLine("Enter the MFA Code sent to your device:");
        string mfaCode = Console.ReadLine();

        AuthFlowResponse mfaResponse = await user.RespondToSmsMfaAuthAsync(new
RespondToSmsMfaRequest()
        {
            SessionID = authResponse.SessionID,
            MfaCode = mfaCode

        }).ConfigureAwait(false);
        accessToken = authResponse.AuthenticationResult.AccessToken;
    }
    else
    {
        Console.WriteLine("Unrecognized authentication challenge.");
        accessToken = "";
        break;
    }
}

if (authResponse.AuthenticationResult != null)
{
    Console.WriteLine("User successfully authenticated.");
}
else
{
    Console.WriteLine("Error in authentication process.");
}
}
```

Utiliser AWS les ressources après l'authentification

Une fois qu'un utilisateur est authentifié à l'aide de la `CognitoAuthentication` bibliothèque, l'étape suivante consiste à lui permettre d'accéder aux AWS ressources appropriées. Pour ce faire, vous devez créer un pool d'identités via la console Amazon Cognito Federated Identities. En spécifiant le groupe d'utilisateurs Amazon Cognito que vous avez créé en tant que fournisseur, à l'aide de son pool et de son ID client, vous pouvez autoriser les utilisateurs de votre groupe d'utilisateurs Amazon Cognito à accéder aux ressources connectées à votre compte. AWS Vous pouvez également spécifier des rôles différents pour permettre aux utilisateurs non authentifiés et authentifiés d'accéder à des ressources différentes. Vous pouvez modifier ces règles dans la console IAM, où vous pouvez ajouter ou supprimer des autorisations dans le champ Action de la stratégie attachée du rôle. Ensuite, en utilisant le groupe d'identités, le groupe d'utilisateurs et les informations utilisateur Amazon Cognito appropriés, vous pouvez passer des appels vers différentes AWS ressources. L'exemple suivant montre un utilisateur authentifié par SRP accédant aux différents compartiments Amazon S3 autorisés par le rôle du pool d'identités associé.

```
public async void GetS3BucketsAsync()
{
    var provider = new AmazonCognitoIdentityProviderClient(new
    AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);

    string password = "userPassword";

    AuthFlowResponse context = await user.StartWithSrpAuthAsync(new
    InitiateSrpAuthRequest()
    {
        Password = password
    }).ConfigureAwait(false);

    CognitoAWSCredentials credentials =
        user.GetCognitoAWSCredentials("identityPoolID", RegionEndpoint.<
    YourIdentityPoolRegion >);

    using (var client = new AmazonS3Client(credentials))
    {
        ListBucketsResponse response =
            await client.ListBucketsAsync(new
            ListBucketsRequest()).ConfigureAwait(false);
    }
}
```



```
        foreach (S3Bucket bucket in response.Buckets)
        {
            Console.WriteLine(bucket.BucketName);
        }
    }
}
```

Plus d'options d'authentification

Outre le SRP et le MFA NewPasswordRequired, CognitoAuthentication la bibliothèque d'extensions facilite le flux d'authentification pour :

- Personnalisé - Initier avec un appel à `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Commencez par un appel à `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Initiez par un appel à `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Initiez par un appel à `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Appelez la méthode appropriée selon le flux souhaité. Continuez ensuite à interroger l'utilisateur avec des stimulations comme elles sont présentées dans les objets `AuthFlowResponse` de chaque appel de méthode. Appelez également la méthode de réponse appropriée, comme `RespondToSmsMfaAuthAsync` pour les stimulations MFA et `RespondToCustomAuthAsync` pour les stimulations personnalisées.

Utilisation des bases de données NoSQL Amazon DynamoDB

Note

Les modèles de programmation décrits dans ces rubriques sont présents à la fois dans .NET Framework et .NET (Core), mais les conventions d'appel diffèrent, qu'elles soient synchrones ou asynchrones.

Il AWS SDK pour .NET prend en charge Amazon DynamoDB, un service de base de données NoSQL rapide proposé par. AWSLe SDK fournit trois modèles de programmation pour communiquer avec DynamoDB : le modèle de bas niveau, le modèle de document et le modèle de persistance des objets.

Les informations suivantes présentent ces modèles et les leurs APIs, fournissent des exemples expliquant comment et quand les utiliser, et vous proposent des liens vers des ressources de programmation DynamoDB supplémentaires dans le. AWS SDK pour .NET

Modèle de bas niveau

Le modèle de programmation de bas niveau englobe les appels directs au service DynamoDB. Vous pouvez accéder à ce modèle via l'espace de noms [Amazon.Dynamo. DBv2](#)

Parmi les trois modèles, le modèle de bas niveau est celui qui nécessite le plus de programmation. Par exemple, vous devez convertir les types de données .NET en leurs équivalents dans DynamoDB. Toutefois, ce modèle donne accès à la plupart des fonctions.

Les exemples suivants montrent comment utiliser le modèle de bas niveau pour créer une table, modifier une table et insérer des éléments dans une table dans DynamoDB.

Création d'une table

L'exemple suivant utilise la méthode `CreateTable` de la classe `AmazonDynamoDBClient` pour créer une table. La méthode `CreateTable` utilise une instance de la classe `CreateTableRequest`, qui contient des caractéristiques telles que des noms d'attribut d'élément requis, une définition de clé primaire et une capacité de débit. La méthode `CreateTable` renvoie une instance de la classe `CreateTableResponse`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

Console.WriteLine("Getting list of tables");
List<string> currentTables = client.ListTables().TableNames;
Console.WriteLine("Number of tables: " + currentTables.Count);
if (!currentTables.Contains("AnimalsInventory"))
{
    var request = new CreateTableRequest
    {
        TableName = "AnimalsInventory",
```

```
        AttributeDefinitions = new List<AttributeDefinition>
    {
        new AttributeDefinition
        {
            AttributeName = "Id",
            // "S" = string, "N" = number, and so on.
            AttributeType = "N"
        },
        new AttributeDefinition
        {
            AttributeName = "Type",
            AttributeType = "S"
        }
    },
    KeySchema = new List<KeySchemaElement>
    {
        new KeySchemaElement
        {
            AttributeName = "Id",
            // "HASH" = hash key, "RANGE" = range key.
            KeyType = "HASH"
        },
        new KeySchemaElement
        {
            AttributeName = "Type",
            KeyType = "RANGE"
        },
    },
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = 10,
        WriteCapacityUnits = 5
    },
};

var response = client.CreateTable(request);

Console.WriteLine("Table created with request ID: " +
    response.ResponseMetadata.RequestId);
}
```

Vérification qu'une table est prête pour modification

Pour pouvoir remplacer ou modifier une table, celle-ci doit être prête pour cette opération. L'exemple suivant montre comment utiliser le modèle de bas niveau pour vérifier qu'une table est prête dans DynamoDB. Dans cet exemple, la table cible à vérifier est référencée par l'intermédiaire de la méthode `DescribeTable` de la classe `AmazonDynamoDBClient`. Toutes les cinq secondes, le code vérifie la valeur de la propriété `TableStatus` de la table. Lorsque le statut est défini sur `ACTIVE`, la table est prête à être modifiée.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var status = "";

do
{
    // Wait 5 seconds before checking (again).
    System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

    try
    {
        var response = client.DescribeTable(new DescribeTableRequest
        {
            TableName = "AnimalsInventory"
        });

        Console.WriteLine("Table = {0}, Status = {1}",
            response.Table.TableName,
            response.Table.TableStatus);

        status = response.Table.TableStatus;
    }
    catch (ResourceNotFoundException)
    {
        // DescribeTable is eventually consistent. So you might
        // get resource not found.
    }
} while (status != TableStatus.ACTIVE);
```

Insertion d'un élément dans une table

Dans l'exemple suivant, vous utilisez le modèle de bas niveau pour insérer deux éléments dans une table de DynamoDB. Chaque élément est inséré par l'intermédiaire de la méthode `PutItem` de la classe `AmazonDynamoDBClient`, en utilisant une instance de la classe `PutItemRequest`. Chacune des deux instances de la classe `PutItemRequest` prend le nom de la table dans laquelle les éléments seront insérés, avec une série de valeurs d'attribut d'élément.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
    TableName = "AnimalsInventory",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "1" } },
        { "Type", new AttributeValue { S = "Dog" } },
        { "Name", new AttributeValue { S = "Fido" } }
    }
};

var request2 = new PutItemRequest
{
    TableName = "AnimalsInventory",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "2" } },
        { "Type", new AttributeValue { S = "Cat" } },
        { "Name", new AttributeValue { S = "Patches" } }
    }
};

client.PutItem(request1);
client.PutItem(request2);
```

Modèle de document

Le modèle de programmation de documents permet de travailler plus facilement avec les données dans DynamoDB. Ce modèle est particulièrement conçu pour l'accès aux tables et aux

éléments des tables. Vous pouvez accéder à ce modèle par le biais d'[Amazon.Dynamo. DBv2 DocumentModel](#) espace de noms.

Comparé au modèle de programmation de bas niveau, le modèle de document est plus facile à coder à partir des données DynamoDB. Par exemple, il n'est pas nécessaire de convertir autant de types de données .NET en leurs équivalents dans DynamoDB. Toutefois, ce modèle ne donne pas accès à autant de fonctions que le modèle de programmation de bas niveau. Par exemple, vous pouvez utiliser ce modèle pour créer, extraire, mettre à jour et supprimer des éléments dans des tables. Toutefois, pour créer les tables, vous devez utiliser le modèle de bas niveau. Comparé au modèle de persistance des objets, ce modèle nécessite davantage de programmation pour stocker, charger et interroger des objets .NET.

Pour plus d'informations sur le modèle de programmation de documents DynamoDB, consultez [.NET : modèle de document dans le](#) manuel Amazon [DynamoDB](#) Developer Guide.

Les sections suivantes fournissent des informations sur la création d'une représentation de la table DynamoDB souhaitée, ainsi que des exemples sur la manière d'utiliser le modèle de document pour insérer des éléments dans des tableaux et obtenir des éléments à partir de tableaux.

Création d'une représentation de la table

Pour effectuer des opérations sur les données à l'aide du modèle de document, vous devez d'abord créer une instance de la `Table` classe qui représente une table spécifique. Il existe deux méthodes principales pour ce faire.

LoadTable méthode

Le premier mécanisme consiste à utiliser l'une des `LoadTable` méthodes statiques de la [Table](#) classe, comme dans l'exemple suivant :

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

Note

Bien que ce mécanisme fonctionne, dans certaines conditions, il peut parfois entraîner une latence supplémentaire ou des blocages en raison des comportements de démarrage à froid et de pool de threads. Pour plus d'informations sur ces comportements, consultez le billet de blog [Modèles d'initialisation DynamoDB améliorés](#) pour le AWS SDK pour .NET

TableBuilder

Un mécanisme alternatif, la [TableBuilder](#) classe, a été introduit dans la [version 3.7.203 du package AWSSDK DBv2 NuGet .Dynamo](#). Ce mécanisme peut remédier aux comportements mentionnés ci-dessus en supprimant certains appels de méthode implicites, en particulier la `DescribeTable` méthode. Ce mécanisme est utilisé de la même manière que dans l'exemple suivant :

```
var client = new AmazonDynamoDBClient();
var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
        DynamoDBEntryType.String, "Message", DynamoDBEntryType.String)
    .Build();
```

Pour plus d'informations sur ce mécanisme alternatif, consultez à nouveau le billet de blog [Modèles d'initialisation DynamoDB améliorés](#) pour le AWS SDK pour .NET

Insertion d'un élément dans un tableau

Dans l'exemple suivant, une réponse est insérée dans le tableau des réponses par le biais de la `PutItemAsync` méthode de la `Table` classe. La méthode `PutItemAsync` prend une instance de la classe `Document` ; la classe `Document` est simplement une collection d'attributs initialisés.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
// by using one of the mechanisms described previously.

// Then, add a reply to the table.
var newReply = new Document();
newReply["Id"] = Guid.NewGuid().ToString();
newReply["ReplyDateTime"] = DateTime.UtcNow;
newReply["PostedBy"] = "Author1";
newReply["Message"] = "Thank you!";

await table.PutItemAsync(newReply);
```

Obtenir un élément d'une table

Dans l'exemple suivant, une réponse est récupérée par le biais de la `GetItemAsync` méthode de la `Table` classe. Pour déterminer la réponse à obtenir, la `GetItemAsync` méthode utilise la clé hash-and-range primaire de la réponse cible.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
// by using one of the mechanisms described previously.

// Then, get a reply from the table
// where "guid" is the hash key and "datetime" is the range key.
var reply = await table.GetItemAsync(guid, datetime);
Console.WriteLine("Id = " + reply["Id"]);
Console.WriteLine("ReplyDateTime = " + reply["ReplyDateTime"]);
Console.WriteLine("PostedBy = " + reply["PostedBy"]);
Console.WriteLine("Message = " + reply["Message"]);
```

L'exemple précédent convertit implicitement les valeurs de table en chaînes pour la `WriteLine` méthode. Vous pouvez effectuer des conversions explicites en utilisant les différentes méthodes « `As [type]` » de la `DynamoDBEntry` classe. Par exemple, vous pouvez convertir explicitement la valeur d'un type `Id` de Primitive données en GUID par le biais de la `AsGuid()` méthode suivante :

```
var guid = reply["Id"].AsGuid();
```

Modèle de persistance des objets

Le modèle de programmation de persistance des objets est spécialement conçu pour le stockage, le chargement et l'interrogation d'objets .NET dans DynamoDB. Vous pouvez accéder à ce modèle par le biais d'[Amazon.Dynamo. DBv2 DataModel](#) espace de noms.

Parmi les trois modèles, le modèle de persistance des objets est le plus facile à utiliser pour coder lorsque vous stockez, chargez ou interrogez des données DynamoDB. Par exemple, vous travaillez directement avec les types de données DynamoDB. Toutefois, ce modèle permet d'accéder uniquement aux opérations qui stockent, chargent et interrogent des objets .NET dans DynamoDB. Par exemple, vous pouvez utiliser ce modèle pour créer, extraire, mettre à jour et supprimer des éléments dans des tables. Toutefois, vous devez tout d'abord créer vos tables à l'aide du modèle de bas niveau, puis utiliser ce modèle pour mapper vos classes .NET avec les tables.

Pour plus d'informations sur le modèle de programmation de persistance des objets DynamoDB, consultez [.NET : modèle de persistance des objets](#) dans le manuel Amazon [DynamoDB](#) Developer Guide.

Les exemples suivants montrent comment définir une classe .NET qui représente un élément DynamoDB, utiliser une instance de la classe .NET pour insérer un élément dans une table DynamoDB et utiliser une instance de la classe .NET pour obtenir un élément de la table.

Définition d'une classe .NET qui représente un élément d'une table

Dans l'exemple suivant de définition de classe, l'`DynamoDBTable` attribut indique le nom de la table, tandis que les `DynamoDBRangeKey` attributs `DynamoDBHashKey` et modélisent la clé hash-and-range primaire de la table. L'`DynamoDBGlobalSecondaryIndexHashKey` attribut est défini de telle sorte qu'une requête de réponses par un auteur spécifique puisse être créée.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

[DynamoDBTable("Reply")]
public class Reply
{
    [DynamoDBHashKey]
    public string Id { get; set; }

    [DynamoDBRangeKey(StoreAsEpoch = false)]
    public DateTime ReplyDateTime { get; set; }

    [DynamoDBGlobalSecondaryIndexHashKey("PostedBy-Message-Index",
        AttributeName = "PostedBy")]
    public string Author { get; set; }

    [DynamoDBGlobalSecondaryIndexRangeKey("PostedBy-Message-Index")]
    public string Message { get; set; }
}
```

Création d'un contexte pour le modèle de persistance des objets

Pour utiliser le modèle de programmation de persistance des objets pour DynamoDB, vous devez créer un contexte qui fournit une connexion à DynamoDB et vous permet d'accéder à des tables, d'effectuer diverses opérations et d'exécuter des requêtes.

Contexte de base

L'exemple suivant montre comment créer le contexte le plus élémentaire.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Contexte lié à `DisableFetchingTableMetadata` la propriété

L'exemple suivant montre comment vous pouvez également définir la `DisableFetchingTableMetadata` propriété de la `DynamoDBContextConfig` classe pour empêcher les appels implicites à la `DescribeTable` méthode.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Si la `DisableFetchingTableMetadata` propriété est définie sur `false` (valeur par défaut), comme indiqué dans le premier exemple, vous pouvez omettre les attributs décrivant la structure de clé et d'index des éléments de table dans la `Reply` classe. Ces attributs seront plutôt déduits par un appel implicite à la `DescribeTable` méthode. `DisableFetchingTableMetadata` est défini sur `true`, comme indiqué dans le deuxième exemple, sur les méthodes du modèle de persistance des objets telles que `SaveAsync` et `QueryAsync` s'appuyant entièrement sur les attributs définis dans la `Reply` classe. Dans ce cas, aucun appel à la `DescribeTable` méthode n'a lieu.

Note

Dans certaines conditions, les appels à la `DescribeTable` méthode peuvent parfois entraîner une latence supplémentaire ou des blocages en raison des comportements de démarrage à froid et de pool de threads. Pour cette raison, il est parfois avantageux d'éviter d'avoir recours à cette méthode.

Pour plus d'informations sur ces comportements, consultez le billet de blog [Modèles d'initialisation DynamoDB améliorés](#) pour le AWS SDK pour .NET

Utilisation d'une instance de la classe .NET pour insérer un élément dans un tableau

Dans cet exemple, un élément est inséré par le biais de la `SaveAsync` méthode de la `DynamoDBContext` classe, qui prend une instance initialisée de la classe .NET qui représente l'élément.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
// examples of which have been described earlier.

// Create an object that represents the new item.
var reply = new Reply()
{
    Id = Guid.NewGuid().ToString(),
    ReplyDateTime = DateTime.UtcNow,
    Author = "Author1",
    Message = "Thank you!"
};

// Insert the item into the table.
await context.SaveAsync<Reply>(reply, new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});
```

Utilisation d'une instance d'une classe .NET pour obtenir des éléments d'une table

Dans cet exemple, une requête est créée pour trouver tous les enregistrements de « Author1 » en utilisant la `QueryAsync` méthode de la `DynamoDBContext` classe. Ensuite, les éléments sont récupérés via la `GetNextSetAsync` méthode de la requête.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
// examples of which have been described earlier.

// Construct a query that finds all replies by a specific author.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
```

```
});

// Display the result.
var set = await query.GetNextSetAsync();
foreach (var item in set)
{
    Console.WriteLine("Id = " + item.Id);
    Console.WriteLine("ReplyDateTime = " + item.ReplyDateTime);
    Console.WriteLine("PostedBy = " + item.Author);
    Console.WriteLine("Message = " + item.Message);
}
```

Informations supplémentaires sur le modèle de persistance des objets

Les exemples et explications présentés ci-dessus incluent parfois une propriété de la `DynamoDBContext` classe appelée `DisableFetchingTableMetadata`. Cette propriété, qui a été introduite dans la [version 3.7.203 du DBv2 NuGet package AWSSDK .Dynamo](#), vous permet d'éviter certaines conditions susceptibles d'entraîner une latence ou des blocages supplémentaires en raison des comportements de démarrage à froid et de pool de threads. Pour plus d'informations, consultez le billet de blog [Modèles d'initialisation DynamoDB améliorés](#) pour le AWS SDK pour .NET

Vous trouverez ci-dessous des informations supplémentaires sur cette propriété.

- Cette propriété peut être définie globalement dans votre `web.config` fichier `app.config` or si vous utilisez .NET Framework.
- Cette propriété peut être définie globalement à l'aide de la [AWSConfigsDynamoDB](#) classe, comme illustré dans l'exemple suivant.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- Dans certains cas, vous ne pouvez pas ajouter d'attributs DynamoDB à une classe .NET, par exemple si la classe est définie dans une dépendance. Dans de tels cas, il est toujours possible de profiter de la `DisableFetchingTableMetadata` propriété. Pour ce faire, utilisez la [TableBuilder](#) classe en plus de la `DisableFetchingTableMetadata` propriété. La `TableBuilder` classe a également été introduite dans la [version 3.7.203 du package AWSSDK .Dynamo. DBv2 NuGet](#)

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
        DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();

// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
    IndexName = "PostedBy-Message-index"
});
```

En savoir plus

Utilisation du AWS SDK pour .NET pour programmer DynamoDB, informations et exemples

- [DynamoDB APIs](#)
- [Lancement de DynamoDB Series](#)
- [DynamoDB Series - Modèle de document](#)
- [DynamoDB Series - Schémas de conversion](#)
- [DynamoDB Series - Modèle de persistance des objets](#)
- [DynamoDB Series - Expressions](#)
- [Utilisation d'expressions avec Amazon DynamoDB et le AWS SDK pour .NET](#)
- [Support JSON dans Amazon DynamoDB](#)

Modèle de bas niveau, informations et exemples

- [Utilisation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Utilisation d'éléments à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Interrogation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Numérisation de tables à l'aide de l'API de AWS SDK pour .NET API de bas niveau](#)
- [Utilisation d'index secondaires locaux à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Utilisation d'index secondaires globaux à l'aide de l'API de AWS SDK pour .NET bas niveau](#)

Modèle de document, informations et exemples

- [Types de données DynamoDB](#)
- [Dynamo DBEntry](#)
- [.NET : Modèle de document](#)

Modèle de persistance des objets, informations et exemples

- [.NET : modèle de persistance des objets](#)

Autres informations utiles

- Consultez [Intégration AWS à .NET Aspire](#) pour plus d'informations sur le développement avec Amazon DynamoDB local via .NET Aspire.

Rubriques

- [Utilisation d'expressions avec Amazon DynamoDB et le AWS SDK pour .NET](#)
- [Support JSON dans Amazon DynamoDB](#)

Utilisation d'expressions avec Amazon DynamoDB et le AWS SDK pour .NET

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Les exemples de code suivants montrent comment utiliser DynamoDB AWS SDK pour .NET pour programmer DynamoDB avec des expressions. Les expressions indiquent les attributs que vous souhaitez lire à partir d'un élément d'une table DynamoDB. Vous utilisez également des expressions lors de l'écriture d'un élément, pour signaler toutes les conditions qui doivent être satisfaites (aussi appelées mise à jour conditionnelle) et pour indiquer comment les attributs doivent être mis à jour. Des exemples de mise à jour remplacent l'attribut par une nouvelle valeur, ou ajoutent de nouvelles données à une liste ou une carte. Pour plus d'informations, consultez [Lecture et écriture d'éléments à l'aide d'expressions](#).

Rubriques

- [Exemples de données](#)
- [Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions](#)
- [Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions](#)
- [Obtenir plusieurs éléments par l'intermédiaire d'expressions et d'autres attributs d'élément](#)
- [Imprimer un élément](#)
- [Créer ou remplacer un élément à l'aide d'expressions](#)
- [Mettre à jour un élément à l'aide d'expressions](#)
- [Supprimer un élément à l'aide d'expressions](#)
- [Plus d'informations](#)

Exemples de données

Les exemples de code présentés dans cette rubrique s'appuient sur les deux exemples d'éléments suivants d'une table DynamoDB nommée. ProductCatalog Ces éléments décrivent des informations sur les entrées de produit dans un catalogue de magasin de cycles fictif. Ces éléments sont basés sur l'exemple fourni dans [Case Study : A ProductCatalog Item](#). Les descripteurs de types de données tels que B00L, L, M, N, NS, S et SS correspondent à ceux de [Format de données JSON](#).

```
{
  "Id": {
    "N": "205"
  },
  "Title": {
    "S": "20-Bicycle 205"
  },
  "Description": {
    "S": "205 description"
  }
}
```

```
{,
  "BicycleType": {
    "S": "Hybrid"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "500"
  },
  "Gender": {
    "S": "B"
  },
  "Color": {
    "SS": [
      "Red",
      "Black"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      }
    ]
  },
},
```



```

    {
      "M": {
        "RearView": {
          "S": "http://example/products/205_rear.jpg"
        }
      }
    },
    {
      "M": {
        "SideView": {
          "S": "http://example/products/205_left_side.jpg"
        }
      }
    }
  ]
},
"ProductReviews": {
  "M": {
    "FiveStar": {
      "SS": [
        "Excellent! Can't recommend it highly enough! Buy it!",
        "Do yourself a favor and buy this."
      ]
    },
    "OneStar": {
      "SS": [
        "Terrible product! Do not buy this."
      ]
    }
  }
},
{
  "Id": {
    "N": "301"
  },
  "Title": {
    "S": "18-Bicycle 301"
  },
  "Description": {
    "S": "301 description"
  },
  "BicycleType": {
    "S": "Road"
  }
}

```

```
},
"Brand": {
  "S": "Brand-Company C"
},
"Price": {
  "N": "185"
},
"Gender": {
  "S": "F"
},
"Color": {
  "SS": [
    "Blue",
    "Silver"
  ]
},
"ProductCategory": {
  "S": "Bike"
},
"InStock": {
  "BOOL": true
},
"QuantityOnHand": {
  "N": "3"
},
"RelatedItems": {
  "NS": [
    "801",
    "822",
    "979"
  ]
},
"Pictures": {
  "L": [
    {
      "M": {
        "FrontView": {
          "S": "http://example/products/301_front.jpg"
        }
      }
    },
    {
      "M": {
        "RearView": {
```

```

        "S": "http://example/products/301_rear.jpg"
    }
}
},
{
    "M": {
        "SideView": {
            "S": "http://example/products/301_left_side.jpg"
        }
    }
}
]
},
"ProductReviews": {
    "M": {
        "FiveStar": {
            "SS": [
                "My daughter really enjoyed this bike!"
            ]
        },
        "ThreeStar": {
            "SS": [
                "This bike was okay, but I would have preferred it in my color.",
                "Fun to ride."
            ]
        }
    }
}
}
}

```

Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions

L'exemple suivant présente la méthode

`Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` et un jeu d'expressions pour obtenir, puis imprimer, l'élément pour lequel `Id` est égal à 205. Seuls les attributs suivants de l'élément sont renvoyés : `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` et `ProductReviews`.

```

// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{

```

```

TableName = "ProductCatalog",
ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
ExpressionAttributeNames = new Dictionary<string, string>
{
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
},
Key = new Dictionary<string, AttributeValue>
{
    { "Id", new AttributeValue { N = "205" } }
},
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);

```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#ri` pour représenter l'attribut `RelatedItems`. L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans [Imprimer un élément](#).

Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions

Les exemples suivants présentent la méthode

`Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` et un jeu d'expressions pour obtenir, puis imprimer, l'élément pour lequel `Id` est égal à 301, mais uniquement si la valeur de `Price` est supérieure à 150. Seuls les attributs suivants de l'élément sont renvoyés : `Id`, `Title` et l'ensemble des attributs `ThreeStar` dans `ProductReviews`.

```

// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new QueryRequest
{
    TableName = "ProductCatalog",
    KeyConditions = new Dictionary<string, Condition>
    {
        { "Id", new Condition()
        {
            ComparisonOperator = ComparisonOperator.EQ,

```

```

        AttributeValueList = new List<AttributeValue>
        {
            new AttributeValue { N = "301" }
        }
    },
    ProjectionExpression = "Id, Title, #pr.ThreeStar",
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#pr", "ProductReviews" },
        { "#p", "Price" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":val", new AttributeValue { N = "150" } }
    },
    FilterExpression = "#p > :val"
};
var response = client.Query(request);

foreach (var item in response.Items)
{
    // Write out the first page of an item's attribute keys and values.
    // PrintItem() is a custom function.
    PrintItem(item);
    Console.WriteLine("====");
}

```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#p` pour représenter l'attribut `Price`. `#pr.ThreeStar` spécifie de retourner uniquement l'attribut `ThreeStar`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:val` pour représenter la valeur `150`. La propriété `FilterExpression` spécifie que `#p` (`Price`) doit être supérieur à `:val` (`150`). L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans [Imprimer un élément](#).

Obtenir plusieurs éléments par l'intermédiaire d'expressions et d'autres attributs d'élément

L'exemple suivant présente la méthode `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` et un jeu d'expressions pour obtenir, puis imprimer, l'ensemble des éléments pour lesquels

ProductCategory est égal à Bike. Seuls les attributs suivants de l'élément sont renvoyés : Id, Title et l'ensemble des attributs dans ProductReviews.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new ScanRequest
{
    TableName = "ProductCatalog",
    ProjectionExpression = "Id, Title, #pr",
    ExpressionAttributeValues = new Dictionary<string,AttributeValue>
    {
        { ":catg", new AttributeValue { S = "Bike" } }
    },
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#pr", "ProductReviews" },
        { "#pc", "ProductCategory" }
    },
    FilterExpression = "#pc = :catg",
};
var response = client.Scan(request);

foreach (var item in response.Items)
{
    // Write out the first page/scan of an item's attribute keys and values.
    // PrintItem() is a custom function.
    PrintItem(item);
    Console.WriteLine("=====");
}
```

Dans l'exemple précédent, la propriété `ProjectionExpression` spécifie l'attribut à renvoyer. La propriété `ExpressionAttributeNames` spécifie l'espace réservé `#pr` pour représenter l'attribut `ProductReviews` et l'espace réservé `#pc` pour représenter l'attribut `ProductCategory`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:catg` pour représenter la valeur `Bike`. La propriété `FilterExpression` spécifie que `#pc` (`ProductCategory`) doit être égal à `:catg` (`Bike`). L'appel à `PrintItem` fait référence à une fonction personnalisée, comme décrit dans [Imprimer un élément](#).

Imprimer un élément

L'exemple suivant présente comment imprimer les attributs et les valeurs d'un élément. Cet exemple est utilisé dans les exemples précédents qui montrent comment [Obtention d'un élément unique à l'aide de sa clé primaire et d'expressions](#), [Obtention de plusieurs éléments à l'aide de la clé primaire de la table et d'expressions](#) et [Obtention de plusieurs éléments à l'aide d'expressions et d'autres attributs d'élément](#).

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
    foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
    {
        Console.Write(kvp.Key + " = ");
        PrintValue(kvp.Value);
    }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
    // Binary attribute value.
    if (value.B != null)
    {
        Console.Write("Binary data");
    }
    // Binary set attribute value.
    else if (value.BS.Count > 0)
    {
        foreach (var bValue in value.BS)
        {
            Console.Write("\n  Binary data");
        }
    }
    // List attribute value.
    else if (value.L.Count > 0)
    {
        foreach (AttributeValue attr in value.L)
        {
            PrintValue(attr);
        }
    }
}
```

```
}
// Map attribute value.
else if (value.M.Count > 0)
{
    Console.Write("\n");
    PrintItem(value.M);
}
// Number attribute value.
else if (value.N != null)
{
    Console.Write(value.N);
}
// Number set attribute value.
else if (value.NS.Count > 0)
{
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
}
// Null attribute value.
else if (value.NULL)
{
    Console.Write("Null");
}
// String attribute value.
else if (value.S != null)
{
    Console.Write(value.S);
}
// String set attribute value.
else if (value.SS.Count > 0)
{
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
}
// Otherwise, boolean value.
else
{
    Console.Write(value.BOOL);
}

Console.Write("\n");
}
```

Dans l'exemple précédent, chaque valeur d'attribut possède plusieurs data-type-specific propriétés qui peuvent être évaluées afin de déterminer le format correct pour imprimer l'attribut. Ces propriétés

incluent B, B00L, BS, L, M, N, NS, NULL, S et SS, qui correspondent à celles du [Format de données JSON](#). Pour des propriétés telles que B, N, NULL et S, si la propriété correspondante n'est pas null, l'attribut est du type de données non null correspondant. Pour les propriétés telles que BSL,M,NS, etSS, si la valeur Count est supérieure à zéro, l'attribut est du type de non-zero-value données correspondant. Si toutes les data-type-specific propriétés de l'attribut sont égales à zéro null ou sont Count égales à zéro, l'attribut correspond au type de B00L données.

Créer ou remplacer un élément à l'aide d'expressions

L'exemple suivant présente la méthode

`Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` et un jeu d'expressions pour mettre à jour l'élément pour lequel Title est 18-Bicycle 301. Si cet élément n'existe pas déjà, un nouvel élément est ajouté.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new PutItemRequest
{
    TableName = "ProductCatalog",
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#title", "Title" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":product", new AttributeValue { S = "18-Bicycle 301" } }
    },
    ConditionExpression = "#title = :product",
    // CreateItemData() is a custom function.
    Item = CreateItemData()
};
client.PutItem(request);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:product` pour représenter la valeur 18-Bicycle 301. La propriété `ConditionExpression` spécifie que `#title` (`Title`) doit être égal à `:product` (18-Bicycle 301). L'appel à `CreateItemData` fait référence à la fonction personnalisée suivante :

```
// using Amazon.DynamoDBv2.Model;

// Provides a sample item that can be added to a table.
public static Dictionary<string, AttributeValue> CreateItemData()
{
    var itemData = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "301" } },
        { "Title", new AttributeValue { S = "18\" Girl's Bike" } },
        { "BicycleType", new AttributeValue { S = "Road" } },
        { "Brand" , new AttributeValue { S = "Brand-Company C" } },
        { "Color", new AttributeValue { SS = new List<string>{ "Blue", "Silver" } } },
        { "Description", new AttributeValue { S = "301 description" } },
        { "Gender", new AttributeValue { S = "F" } },
        { "InStock", new AttributeValue { BOOL = true } },
        { "Pictures", new AttributeValue { L = new List<AttributeValue>{
            { new AttributeValue { M = new Dictionary<string,AttributeValue>{
                { "FrontView", new AttributeValue { S = "http://example/
products/301_front.jpg" } } } } },
            { new AttributeValue { M = new Dictionary<string,AttributeValue>{
                { "RearView", new AttributeValue {S = "http://example/
products/301_rear.jpg" } } } } },
            { new AttributeValue { M = new Dictionary<string,AttributeValue>{
                { "SideView", new AttributeValue { S = "http://example/
products/301_left_side.jpg" } } } } }
        } } },
        { "Price", new AttributeValue { N = "185" } },
        { "ProductCategory", new AttributeValue { S = "Bike" } },
        { "ProductReviews", new AttributeValue { M = new Dictionary<string,AttributeValue>{
            { "FiveStar", new AttributeValue { SS = new List<string>{
                "My daughter really enjoyed this bike!" } } },
            { "OneStar", new AttributeValue { SS = new List<string>{
                "Fun to ride.",
                "This bike was okay, but I would have preferred it in my color." } } }
        } } },
        { "QuantityOnHand", new AttributeValue { N = "3" } },
        { "RelatedItems", new AttributeValue { NS = new List<string>{ "979", "822",
"801" } } }
    };

    return itemData;
}
```

Dans l'exemple précédent, un exemple d'élément avec des exemples de données est renvoyé au mandataire. Une série d'attributs et de valeurs correspondantes est construite, à l'aide des types de données tels que B00L, L, M, N, NS, S et SS, qui correspondent à ceux du [format de données JSON](#).

Mettre à jour un élément à l'aide d'expressions

L'exemple suivant présente la méthode

`Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` et un jeu d'expressions pour modifier `Title` en `18" Girl's Bike` pour l'élément pour lequel `Id` est égal à `301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
    TableName = "ProductCatalog",
    Key = new Dictionary<string,AttributeValue>
    {
        { "Id", new AttributeValue { N = "301" } }
    },
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#title", "Title" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
    },
    UpdateExpression = "SET #title = :newproduct"
};
client.UpdateItem(request);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:newproduct` pour représenter la valeur `18" Girl's Bike`. La propriété `UpdateExpression` spécifie de modifier `#title (Title)` en `:newproduct (18" Girl's Bike)`.

Supprimer un élément à l'aide d'expressions

L'exemple suivant présente la méthode

`Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` et un jeu d'expressions pour supprimer les éléments pour lesquels `Id` est égal à `301` et `Title` est égal à `18-Bicycle 301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
    TableName = "ProductCatalog",
    Key = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "301" } }
    },
    ExpressionAttributeNames = new Dictionary<string, string>
    {
        { "#title", "Title" }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":product", new AttributeValue { S = "18-Bicycle 301" } }
    },
    ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

Dans l'exemple précédent, la propriété `ExpressionAttributeNames` spécifie l'espace réservé `#title` pour représenter l'attribut `Title`. La propriété `ExpressionAttributeValues` spécifie l'espace réservé `:product` pour représenter la valeur `18-Bicycle 301`. La propriété `ConditionExpression` spécifie que `#title` (`Title`) doit être égal à `:product` (`18-Bicycle 301`).

Plus d'informations

Pour plus d'informations et des exemples de code, consultez :

- [DynamoDB Series - Expressions](#)
- [Accès à des attributs d'élément avec des expressions de projections](#)

- [Utilisation d'espaces réservés pour valeurs et noms d'attributs](#)
- [Spécification de conditions avec des expressions de condition](#)
- [Modification d'éléments et d'attributs avec des expressions de mises à jour](#)
- [Utilisation d'éléments à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Interrogation de tables à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Numérisation de tables à l'aide de l' API de AWS SDK pour .NET API de bas niveau](#)
- [Utilisation d'index secondaires locaux à l'aide de l'API de AWS SDK pour .NET bas niveau](#)
- [Utilisation d'index secondaires globaux à l'aide de l'API de AWS SDK pour .NET bas niveau](#)

Support JSON dans Amazon DynamoDB

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Il AWS SDK pour .NET prend en charge les données JSON lorsque vous travaillez avec Amazon DynamoDB. Cela vous permet d'obtenir plus facilement des données au format JSON à partir de tables DynamoDB et d'y insérer des documents JSON.

Rubriques

- [Obtenir des données à partir d'une table DynamoDB au format JSON](#)
- [Insérer des données au format JSON dans une table DynamoDB](#)
- [Conversions des types de données DynamoDB en JSON](#)
- [Plus d'informations](#)

Obtenir des données à partir d'une table DynamoDB au format JSON

L'exemple suivant montre comment obtenir des données d'une table DynamoDB au format JSON :

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
```

```
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
// {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
// {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
// }
```

Dans l'exemple précédent, la méthode `ToJson` de la classe `Document` convertit un élément de la table en chaîne au format JSON. La méthode `GetItem` de la classe `Table` permet de récupérer cet élément. Pour déterminer l'élément à obtenir, dans cet exemple, la `GetItem` méthode utilise la clé hash-and-range primaire de l'élément cible. Pour déterminer la table à partir de laquelle obtenir l'élément, la `LoadTable` méthode de la `Table` classe utilise une instance de la `AmazonDynamoDBClient` classe et le nom de la table cible dans `DynamoDB`.

Insérer des données au format JSON dans une table `DynamoDB`

L'exemple suivant montre comment utiliser le format JSON pour insérer un élément dans une table `DynamoDB` :

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var jsonText = "{\"Id\":6,\"Type\":\"Bird\",\"Name\":\"Tweety\"}";
var item = Document.FromJson(jsonText);

table.PutItem(item);
```

Dans l'exemple précédent, la méthode `FromJson` de la classe `Document` convertit une chaîne au format JSON en élément. L'élément est inséré dans la table par l'intermédiaire de la méthode

PutItem de la classe Table, qui utilise l'instance de la classe Document contenant l'élément. Pour déterminer la table dans laquelle insérer l'élément, la LoadTable méthode de la Table classe est appelée, en spécifiant une instance de la AmazonDynamoDBClient classe et le nom de la table cible dans DynamoDB.

Conversions des types de données DynamoDB en JSON

Chaque fois que vous appelez la ToJson méthode de la Document classe, puis que vous appelez la FromJson méthode pour reconvertir les données JSON en instance de Document classe, certains types de données DynamoDB ne sont pas convertis comme prévu. En particulier :

- Les ensembles DynamoDB (SSles typesNS, BS et) seront convertis en tableaux JSON.
- Les scalaires et ensembles binaires DynamoDB (Bles types BS et) seront convertis en chaînes JSON codées en base64 ou en listes de chaînes.

Dans ce scénario, il convient d'appeler la méthode DecodeBase64Attributes de la classe Document pour remplacer les données JSON codées en base64 par la représentation binaire correcte. L'exemple suivant remplace par la représentation binaire correcte un attribut d'élément binary scalar codé en base64 dans une instance d'une classe Document nommée Picture. Cet exemple procède de même pour un attribut d'élément binary set codé en base64 dans la même instance de la classe Document nommée RelatedPictures :

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Plus d'informations

Pour plus d'informations et des exemples de programmation de JSON avec DynamoDB avec AWS SDK pour .NET le, voir :

- [Support DynamoDB pour JSON](#)
- [Mise à jour d'Amazon DynamoDB : JSON, offre gratuite élargie, dimensionnement flexible, éléments plus volumineux](#)

Utilisation avec Amazon EC2

Il AWS SDK pour .NET prend en charge [Amazon EC2](#), un service Web qui fournit une capacité de calcul redimensionnable. Vous utilisez cette capacité de calcul pour créer et héberger vos systèmes logiciels.

APIs

AWS SDK pour .NET II fournit des API pour les clients Amazon EC2. Les API vous permettent de travailler avec des fonctionnalités EC2 telles que les groupes de sécurité et les paires de clés. Les API vous permettent également de contrôler les instances Amazon EC2. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la [référence de l'AWS SDK pour .NET API](#) (et faites défiler la page jusqu'à « Amazon.ec2 »).

Les Amazon EC2 APIs sont fournis par le package [AWSSDK NuGet .EC2](#).

Conditions préalables

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

À propos des exemples

Les exemples présentés dans cette section vous montrent comment travailler avec les clients Amazon EC2 et gérer les instances Amazon EC2.

Le [didacticiel sur les instances Spot EC2](#) explique comment demander des instances Spot Amazon EC2. Les instances Spot vous permettent d'accéder à de la capacité EC2 inutilisée à un prix inférieur au prix à la demande.

Rubriques

- [Travailler avec des groupes de sécurité dans Amazon EC2](#)
- [Utilisation des paires de clés Amazon EC2](#)
- [Afficher vos régions et zones de disponibilité Amazon EC2](#)
- [Utilisation des instances Amazon EC2](#)
- [Tutoriel sur les instances Spot Amazon EC2](#)

Travailler avec des groupes de sécurité dans Amazon EC2

Dans Amazon EC2, un groupe de sécurité agit comme un pare-feu virtuel qui contrôle le trafic réseau pour une ou plusieurs instances EC2. Par défaut, EC2 associe vos instances à un groupe de sécurité qui n'autorise aucun trafic entrant. Vous pouvez créer un groupe de sécurité qui autorise vos instances EC2 à accepter un certain trafic. Par exemple, si vous devez vous connecter à une instance Windows EC2, vous devez configurer le groupe de sécurité afin d'autoriser le trafic RDP.

Pour en savoir plus sur les groupes de sécurité, consultez les groupes de [sécurité Amazon EC2 dans le guide](#) de l'utilisateur [Amazon EC2](#).

 Warning

EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog [EC2-Classic Networking is Retiring — Here's How to Prepare](#).

Pour plus d'informations sur les APIs et les prérequis, consultez la section parent ([Utilisation avec Amazon EC2](#)).

Rubriques

- [Énumération des groupes de sécurité](#)
- [Création de groupes de sécurité](#)
- [Mise à jour des groupes de sécurité](#)

Énumération des groupes de sécurité

Cet exemple montre comment utiliser le pour AWS SDK pour .NET énumérer les groupes de sécurité. Si vous fournissez un [Amazon Virtual Private Cloud](#) ID, l'application énumère les groupes de sécurité pour ce VPC en particulier. Dans le cas contraire, l'application affiche simplement la liste de tous les groupes de sécurité disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Énumérer les groupes de sécurité](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Énumérer les groupes de sécurité

L'extrait suivant répertorie vos groupes de sécurité. Il énumère tous les groupes ou les groupes d'un VPC en particulier s'il en existe un.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to enumerate the security groups
private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
{
    // A request object, in case we need it.
    var request = new DescribeSecurityGroupsRequest();

    // Put together the properties, if needed
    if(!string.IsNullOrEmpty(vpcID))
    {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"\\nGetting security groups for VPC {vpcID}...\\n");
        request.Filters.Add(new Filter
        {
            Name = "vpc-id",
            Values = new List<string>() { vpcID }
        });
    }

    // Get the list of security groups
    DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

    // Display the list of security groups.
    foreach (SecurityGroup item in response.SecurityGroups)
    {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\\tGroupId: " + item.GroupId);
        Console.WriteLine("\\tGroupName: " + item.GroupName);
        Console.WriteLine("\\tVpcId: " + item.VpcId);
        Console.WriteLine();
    }
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

[Filtre](#) de classe

Classe [SecurityGroup](#)

Le code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2EnumerateSecGroups
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line
            string vpcID = string.Empty;
            if(args.Length == 0)
            {
                Console.WriteLine("\nEC2EnumerateSecGroups [vpc_id]");
                Console.WriteLine("  vpc_id - The ID of the VPC for which you want to see
security groups.");
                Console.WriteLine("\nSince you specified no arguments, showing all available
security groups.");
            }
            else
```

```

    {
        vpcID = args[0];
    }

    if(vpcID.StartsWith("vpc-") || string.IsNullOrEmpty(vpcID))
    {
        // Create an EC2 client object
        var ec2Client = new AmazonEC2Client();

        // Enumerate the security groups
        await EnumerateGroups(ec2Client, vpcID);
    }
    else
    {
        Console.WriteLine("Could not find a valid VPC ID in the command-line arguments:");
        Console.WriteLine($"{args[0]}");
    }
}

//
// Method to enumerate the security groups
private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
{
    // A request object, in case we need it.
    var request = new DescribeSecurityGroupsRequest();

    // Put together the properties, if needed
    if(!string.IsNullOrEmpty(vpcID))
    {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"{ "\nGetting security groups for VPC {vpcID}...\n");
        request.Filters.Add(new Filter
        {
            Name = "vpc-id",
            Values = new List<string>() { vpcID }
        });
    }

    // Get the list of security groups
    DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

```

```
// Display the list of security groups.
foreach (SecurityGroup item in response.SecurityGroups)
{
    Console.WriteLine("Security group: " + item.GroupId);
    Console.WriteLine("\tGroupId: " + item.GroupId);
    Console.WriteLine("\tGroupName: " + item.GroupName);
    Console.WriteLine("\tVpcId: " + item.VpcId);
    Console.WriteLine();
}
}
```

Considérations supplémentaires

- Notez que dans le cas d'un VPC, le filtre est construit avec la Name partie de la paire nom-valeur définie sur « vpc-id ». Ce nom provient de la description de la `Filters` propriété de la [DescribeSecurityGroupsRequest](#) classe.
- Pour obtenir la liste complète de vos groupes de sécurité, vous pouvez également utiliser [DescribeSecurityGroupsAsync](#) sans paramètres.
- Vous pouvez vérifier les résultats en consultant la liste des groupes de sécurité dans la console [Amazon EC2](#).

Création de groupes de sécurité

Cet exemple montre comment utiliser le AWS SDK pour .NET pour créer un groupe de sécurité. Vous pouvez fournir l'ID d'un VPC existant afin de créer un groupe de sécurité pour EC2 dans un VPC. Si vous ne fournissez pas un tel identifiant, le nouveau groupe de sécurité sera destiné à EC2-Classique si votre AWS compte le permet.

Si vous ne fournissez pas d'identifiant VPC et que votre AWS compte ne prend pas en charge EC2-Classique, le nouveau groupe de sécurité appartiendra au VPC par défaut de votre compte.

⚠ Warning

EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog [EC2-Classic Networking is Retiring — Here's How to Prepare](#).

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Rechercher des groupes de sécurité existants](#)
- [Création d'un groupe de sécurité](#)
- [Code complet](#)

Rechercher des groupes de sécurité existants

L'extrait suivant recherche les groupes de sécurité existants portant le nom donné dans le VPC donné.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to determine if a security group with the specified name  
// already exists in the VPC  
private static async Task<List<SecurityGroup>> FindSecurityGroups(  
    IAmazonEC2 ec2Client, string groupName, string vpcID)  
{  
    var request = new DescribeSecurityGroupsRequest();  
    request.Filters.Add(new Filter{  
        Name = "group-name",  
        Values = new List<string>() { groupName }  
    });  
    if(!string.IsNullOrEmpty(vpcID))  
        request.Filters.Add(new Filter{  
            Name = "vpc-id",  
            Values = new List<string>() { vpcID }  
        });  
  
    var response = await ec2Client.DescribeSecurityGroupsAsync(request);
```

```
return response.SecurityGroups;
}
```

Création d'un groupe de sécurité

L'extrait suivant crée un nouveau groupe de sécurité si aucun groupe portant ce nom n'existe dans le VPC donné. Si aucun VPC n'est indiqué et qu'un ou plusieurs groupes portant ce nom existent, l'extrait renvoie simplement la liste des groupes.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to create a new security group (either EC2-Classic or EC2-VPC)
// If vpcID is empty, the security group will be for EC2-Classic
private static async Task<List<SecurityGroup>> CreateSecurityGroup(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    // See if one or more security groups with that name
    // already exist in the given VPC. If so, return the list of them.
    var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
    if (securityGroups.Count > 0)
    {
        Console.WriteLine(
            $"One or more security groups with name {groupName} already exist.\n");
        return securityGroups;
    }

    // If the security group doesn't already exist, create it.
    var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
    };
    if(string.IsNullOrEmpty(vpcID))
    {
        createRequest.Description = "My .NET example security group for EC2-Classic";
    }
    else
    {
        createRequest.VpcId = vpcID;
        createRequest.Description = "My .NET example security group for EC2-VPC";
    }
    CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);
}
```

```
// Return the new security group
DescribeSecurityGroupsResponse describeResponse =
    await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
        GroupIds = new List<string>() { createResponse.GroupId }
    });
return describeResponse.SecurityGroups;
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [CreateSecurityGroupRequest](#)

Classe [CreateSecurityGroupResponse](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

[Filtre](#) de classe

Classe [SecurityGroup](#)

Le code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
```



```

using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
    // = = = = =
    // Class to create a security group
    class Program
    {
        private const int MaxArgs = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }
            if(parsedArgs.Count > MaxArgs)
                CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Get the application arguments from the parsed list
            var groupName = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-name");
            var vpcID = CommandLine.GetArgument(parsedArgs, null, "-v", "--vpc-id");
            if(string.IsNullOrEmpty(groupName))
                CommandLine.ErrorExit("\nYou must supply a name for the new group." +
                    "\nRun the command with no arguments to see help.");
            if(!string.IsNullOrEmpty(vpcID) && !vpcID.StartsWith("vpc-"))
                CommandLine.ErrorExit($"Not a valid VPC ID: {vpcID}");

            // groupName has a value and vpcID either has a value or is null (which is fine)
            // Create the new security group and display information about it
            var securityGroups =
                await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
            Console.WriteLine("Information about the security group(s):");
            foreach(var group in securityGroups)
            {
                Console.WriteLine($"GroupName: {group.GroupName}");
                Console.WriteLine($"GroupId: {group.GroupId}");
                Console.WriteLine($"Description: {group.Description}");
            }
        }
    }
}

```

```

        Console.WriteLine($"VpcId (if any): {group.VpcId}");
    }
}

//
// Method to create a new security group (either EC2-Classic or EC2-VPC)
// If vpcID is empty, the security group will be for EC2-Classic
private static async Task<List<SecurityGroup>> CreateSecurityGroup(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    // See if one or more security groups with that name
    // already exist in the given VPC. If so, return the list of them.
    var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
    if (securityGroups.Count > 0)
    {
        Console.WriteLine(
            $"\\nOne or more security groups with name {groupName} already exist.\\n");
        return securityGroups;
    }

    // If the security group doesn't already exist, create it.
    var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
    };
    if(string.IsNullOrEmpty(vpcID))
    {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
    }
    else
    {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
    }
    CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

    // Return the new security group
    DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
            GroupIds = new List<string>() { createResponse.GroupId }
        });
}

```

```

    return describeResponse.SecurityGroups;
}

//
// Method to determine if a security group with the specified name
// already exists in the VPC
private static async Task<List<SecurityGroup>> FindSecurityGroups(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    var request = new DescribeSecurityGroupsRequest();
    request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
    });
    if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
            Name = "vpc-id",
            Values = new List<string>() { vpcID }
        });

    var response = await ec2Client.DescribeSecurityGroupsAsync(request);
    return response.SecurityGroups;
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n      If vpc-id isn't present, the security group will be" +
        "\n      for EC2-Classic (if your AWS account supports this)" +
        "\n      or will use the default VCP for EC2-VPC.");
}
}

```

```
// = = = = =
= = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];
                var value = key;

                // Check to see if there's a value that goes with this option?
                if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
                parsedArgs.Add(key, value);
            }

            // If the first argument in this iteration doesn't start with a dash, it's a
value
            else
            {
                parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
                n++;
            }
        }
    }
}
```

```

    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}
}

```

Mise à jour des groupes de sécurité

Cet exemple montre comment utiliser le pour AWS SDK pour .NET ajouter une règle à un groupe de sécurité. En particulier, l'exemple ajoute une règle pour autoriser le trafic entrant sur un port TCP donné, qui peut être utilisée, par exemple, pour les connexions à distance à une instance EC2. L'application prend l'identifiant d'un groupe de sécurité existant, une adresse IP (ou plage d'adresses) au format CIDR et éventuellement un numéro de port TCP. Il ajoute ensuite une règle entrante au groupe de sécurité donné.

Note

Pour utiliser cet exemple, vous avez besoin d'une adresse IP (ou d'une plage d'adresses) au format CIDR. Consultez la section [Considérations supplémentaires](#) à la fin de cette rubrique pour connaître les méthodes permettant d'obtenir l'adresse IP de votre ordinateur local.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Ajouter une règle de trafic entrant](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Ajouter une règle de trafic entrant

L'extrait suivant ajoute une règle entrante à un groupe de sécurité pour une adresse IP (ou plage) et un port TCP particuliers.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
    IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
    // Create an object to hold the request information for the rule.
    // It uses an IpPermission object to hold the IP information for the rule.
    var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID;
        ingressRequest.IpPermissions.Add(new IpPermission{
            IpProtocol = "tcp",
            FromPort = port,
            ToPort = port,
            Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
        });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
```

```

        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
        Console.WriteLine($"\\nNew RDP rule was written in {groupID} for {ipAddress}.");
        Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }

```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [AuthorizeSecurityGroupIngressRequest](#)

Classe [AuthorizeSecurityGroupIngressResponse](#)

Classe [IpPermission](#)

Classe [IpRange](#)

Le code

```

using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2AddRuleForRDP
{
    // = = = = =
    = = =

```

```
// Class to add a rule that allows inbound traffic on TCP a port
class Program
{
    private const int DefaultPort = 3389;

    static async Task Main(string[] args)
    {
        // Parse the command line and show help if necessary
        var parsedArgs = CommandLine.Parse(args);
        if(parsedArgs.Count == 0)
        {
            PrintHelp();
            return;
        }

        // Get the application arguments from the parsed list
        var groupId = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
        var ipAddress = CommandLine.GetArgument(parsedArgs, null, "-i", "--ip-address");
        var portStr = CommandLine.GetArgument(parsedArgs, DefaultPort.ToString(), "-p",
"--port");
        if(string.IsNullOrEmpty(ipAddress))
            CommandLine.ErrorExit("\nYou must supply an IP address in CIDR format.");
        if(string.IsNullOrEmpty(groupId) || !groupId.StartsWith("sg-"))
            CommandLine.ErrorExit("\nThe ID for a security group is missing or
incorrect.");
        if(int.Parse(portStr) == 0)
            CommandLine.ErrorExit($"The given TCP port number, {portStr}, isn't
allowed.");

        // Add a rule to the given security group that allows
        // inbound traffic on a TCP port
        await AddIngressRule(
            new AmazonEC2Client(), groupId, ipAddress, int.Parse(portStr));
    }

    //
    // Method that adds a TCP ingress rule to a security group
    private static async Task AddIngressRule(
        IAmazonEC2 eC2Client, string groupId, string ipAddress, int port)
    {
        // Create an object to hold the request information for the rule.
        // It uses an IpPermission object to hold the IP information for the rule.
        var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
```



```

        GroupId = groupId};
    ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
    });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
        await e2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"\\nNew RDP rule was written in {groupId} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\\nUsage: EC2AddRuleForRDP -g <group-id> -i <ip-address> [-p <port>]" +
        "\\n  -g, --group-id: The ID of the security group to which you want to add the
inbound rule." +
        "\\n  -i, --ip-address: An IP address or address range in CIDR format." +
        "\\n  -p, --port: The TCP port number. Defaults to 3389.");
}

}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:

```

```

// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.

```

```
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
```

Considérations supplémentaires

- Si vous ne fournissez pas de numéro de port, l'application utilise par défaut le port 3389. Il s'agit du port pour Windows RDP, qui vous permet de vous connecter à une instance EC2 exécutant Windows. Si vous lancez une instance EC2 exécutant Linux, vous pouvez utiliser le port TCP 22 (SSH) à la place.
- Notez que l'exemple prend la valeur `IpProtocol` « tcp ». Les valeurs de `se IpProtocol` trouvent dans la description de la `IpProtocol` propriété de la [IpPermission](#) classe.
- Vous aurez peut-être besoin de l'adresse IP de votre ordinateur local lorsque vous utiliserez cet exemple. Voici certains des moyens par lesquels vous pouvez obtenir l'adresse.
 - Si votre ordinateur local (à partir duquel vous allez vous connecter à votre instance EC2) possède une adresse IP publique statique, vous pouvez utiliser un service pour obtenir cette adresse. L'un de ces services est <http://checkip.amazonaws.com/>. Pour en savoir plus sur l'autorisation du trafic entrant, consultez les sections [Ajouter des règles à un groupe de sécurité et Règles de groupe de sécurité pour différents cas d'utilisation](#) dans le guide de l'utilisateur [Amazon EC2](#).

- Une autre méthode pour obtenir l'adresse IP de votre ordinateur local consiste à utiliser la console [Amazon EC2](#).

Sélectionnez l'un de vos groupes de sécurité, sélectionnez l'onglet Règles entrantes, puis choisissez Modifier les règles entrantes. Dans une règle entrante, ouvrez le menu déroulant dans la colonne Source et choisissez Mon adresse IP pour voir l'adresse IP de votre ordinateur local au format CIDR. Assurez-vous d'annuler l'opération.

- Vous pouvez vérifier les résultats de cet exemple en examinant la liste des groupes de sécurité dans la console [Amazon EC2](#).

Utilisation des paires de clés Amazon EC2

Amazon EC2 utilise le chiffrement à clé publique pour chiffrer et déchiffrer les informations de connexion. Le chiffrement à clé publique utilise une clé publique pour chiffrer les données, puis le destinataire utilise la clé privée pour déchiffrer les données. La clé publique et la clé privée constituent une paire de clés. Si vous souhaitez vous connecter à une instance EC2, vous devez spécifier une paire de clés lorsque vous la lancez, puis fournir la clé privée de la paire lorsque vous vous y connectez.

Lorsque vous lancez une instance EC2, vous pouvez créer une paire de clés pour celle-ci ou utiliser une paire de clés que vous avez déjà utilisée lors du lancement d'autres instances. Pour en savoir plus sur les paires de clés Amazon EC2, consultez la section [Utilisation des paires de clés Amazon EC2](#) dans le guide de l'utilisateur Amazon [EC2](#).

Pour plus d'informations sur les APIs et les prérequis, consultez la section parent ([Utilisation avec Amazon EC2](#)).

Rubriques

- [Création et affichage de paires de clés](#)
- [Supprimer des paires de clés](#)

Création et affichage de paires de clés

Cet exemple montre comment utiliser le AWS SDK pour .NET pour créer une paire de clés. L'application prend le nom de la nouvelle paire de clés et le nom d'un fichier PEM (avec une extension « .pem »). Il crée la paire de clés, écrit la clé privée dans le fichier PEM, puis affiche

toutes les paires de clés disponibles. Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement toutes les paires de clés disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Créer la paire de clés](#)
- [Afficher les paires de clés disponibles](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Créer la paire de clés

L'extrait suivant crée une paire de clés, puis stocke la clé privée dans le fichier PEM donné.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to create a key pair and save the key material in a PEM file  
private static async Task CreateKeyPair(  
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)  
{  
    // Create the key pair  
    CreateKeyPairResponse response =  
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{  
            KeyName = keyPairName  
        });  
    Console.WriteLine($"\\nCreated new key pair: {response.KeyPair.KeyName}");  
  
    // Save the private key in a PEM file  
    using (var s = new FileStream(pemFileName, FileMode.Create))  
    using (var writer = new StreamWriter(s))  
    {  
        writer.WriteLine(response.KeyPair.KeyMaterial);  
    }  
}
```

Afficher les paires de clés disponibles

L'extrait suivant affiche la liste des paires de clés disponibles.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [CreateKeyPairRequest](#)

Classe [CreateKeyPairResponse](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyValuePair](#)

Le code

```
using System;  
using System.Threading.Tasks;  
using System.IO;
```

```

using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
    // = = = = =
    // Class to create and store a key pair
    class Program
    {
        static async Task Main(string[] args)
        {
            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                // In the case of no command-line arguments,
                // just show help and the existing key pairs
                PrintHelp();
                Console.WriteLine("\nNo arguments specified.");
                Console.Write(
                    "Do you want to see a list of the existing key pairs? ((y) or n): ");
                string response = Console.ReadLine();
                if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
                    await EnumerateKeyPairs(ec2Client);
                return;
            }

            // Get the application arguments from the parsed list
            string keyPairName =
                CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
            string pemFileName =
                CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
            if(string.IsNullOrEmpty(keyPairName))
                CommandLine.ErrorExit("\nNo key pair name specified." +
                    "\nRun the command with no arguments to see help.");
            if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
                CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
                    "\nRun the command with no arguments to see help.");
        }
    }
}

```

```

    // Create the key pair
    await CreateKeyPair(ec2Client, keyPairName, pemFileName);
    await EnumerateKeyPairs(ec2Client);
}

//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
    // Create the key pair
    CreateKeyPairResponse response =
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
            KeyName = keyPairName
        });
    Console.WriteLine($"Created new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
        writer.WriteLine(response.KeyPair.KeyMaterial);
    }
}

//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename> " +

```



```

        "\n -k, --keypair-name: The name you want to assign to the key pair." +
        "\n -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
    }
}

// =====
//
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];
                var value = key;

                // Check to see if there's a value that goes with this option?
                if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
                parsedArgs.Add(key, value);
            }

```

```

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}

```

Considérations supplémentaires

- Après avoir exécuté l'exemple, vous pouvez voir la nouvelle paire de clés dans la console [Amazon EC2](#).
- Lorsque vous créez une paire de clés, vous devez enregistrer la clé privée renvoyée car vous ne pourrez pas la récupérer ultérieurement.

Supprimer des paires de clés

Cet exemple montre comment utiliser le AWS SDK pour .NET pour supprimer une paire de clés. L'application prend le nom d'une paire de clés. Il supprime la paire de clés, puis affiche toutes les paires de clés disponibles. Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement toutes les paires de clés disponibles.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Suppression de la paire de clés](#)
- [Afficher les paires de clés disponibles](#)
- [Code complet](#)

Suppression de la paire de clés

L'extrait de code suivant supprime une paire de clés.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to delete a key pair  
private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)  
{  
    await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{  
        KeyName = keyName});  
    Console.WriteLine($"\\nKey pair {keyName} has been deleted (if it existed).");  
}
```

Afficher les paires de clés disponibles

L'extrait suivant affiche la liste des paires de clés disponibles.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [DeleteKeyPairRequest](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyValuePair](#)

Le code

```
using System;
```

```
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2DeleteKeyPair
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            if(args.Length == 1)
            {
                // Delete a key pair (if it exists)
                await DeleteKeyPair(ec2Client, args[0]);

                // Display the key pairs that are left
                await EnumerateKeyPairs(ec2Client);
            }
            else
            {
                Console.WriteLine("\nUsage: EC2DeleteKeyPair keypair-name");
                Console.WriteLine("  keypair-name - The name of the key pair you want to
delete.");
                Console.WriteLine("\nNo arguments specified.");
                Console.Write(
                    "Do you want to see a list of the existing key pairs? ((y) or n): ");
                string response = Console.ReadLine();
                if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
                    await EnumerateKeyPairs(ec2Client);
            }
        }

        //
        // Method to delete a key pair
        private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)
        {
            await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{
                KeyName = keyName});
            Console.WriteLine($"Key pair {keyName} has been deleted (if it existed).");
        }
    }
}
```

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}  
}
```

Afficher vos régions et zones de disponibilité Amazon EC2

Amazon EC2 est hébergé sur plusieurs sites dans le monde entier. Ces emplacements sont composés de régions et de zones de disponibilité. Chaque région est une zone géographique distincte qui compte plusieurs emplacements isolés appelés zones de disponibilité.

Pour en savoir plus sur les régions et les zones de disponibilité, consultez [Régions et zones](#) dans le guide de l'[utilisateur Amazon EC2](#).

Cet exemple vous montre comment utiliser le pour AWS SDK pour .NET obtenir des informations sur les régions et les zones de disponibilité associées à un client EC2. L'application affiche les listes des régions et des zones de disponibilité disponibles pour un client EC2.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [DescribeAvailabilityZonesResponse](#)

Classe [DescribeRegionsResponse](#)

Classe [AvailabilityZone](#)

[Région](#) de classe

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2RegionsAndZones
{
    class Program
    {
        static async Task Main(string[] args)
        {
            Console.WriteLine(
                "Finding the Regions and Availability Zones available to an EC2 client...");

            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            // Display the Regions and Availability Zones
            await DescribeRegions(ec2Client);
            await DescribeAvailabilityZones(ec2Client);
        }

        //
        // Method to display Regions
        private static async Task DescribeRegions(IAmazonEC2 ec2Client)
        {
            Console.WriteLine("\nRegions that are enabled for the EC2 client:");
            DescribeRegionsResponse response = await ec2Client.DescribeRegionsAsync();
            foreach (Region region in response.Regions)
            {
                Console.WriteLine(region.RegionName);
            }
        }

        //
        // Method to display Availability Zones
```

```
private static async Task DescribeAvailabilityZones(IAmazonEC2 ec2Client)
{
    Console.WriteLine("\nAvailability Zones for the EC2 client's region:");
    DescribeAvailabilityZonesResponse response =
        await ec2Client.DescribeAvailabilityZonesAsync();
    foreach (AvailabilityZone az in response.AvailabilityZones)
        Console.WriteLine(az.ZoneName);
    }
}
```

Utilisation des instances Amazon EC2

Vous pouvez utiliser le AWS SDK pour .NET pour contrôler les instances Amazon EC2 à l'aide d'opérations telles que la création, le démarrage et la fin. Les rubriques de cette section fournissent quelques exemples de la manière de procéder. Pour en savoir plus sur les instances EC2, consultez les instances [Amazon EC2](#) dans le guide de l'utilisateur [Amazon EC2](#).

Pour plus d'informations sur le APIs et les prérequis, consultez la section parent ([Utilisation avec Amazon EC2](#)).

Rubriques

- [Lancement d'une instance Amazon EC2](#)
- [Interruption d'une instance Amazon EC2](#)

Lancement d'une instance Amazon EC2

Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour lancer une ou plusieurs instances Amazon EC2 configurées de manière identique à partir de la même Amazon Machine Image (AMI). À l'aide de [plusieurs entrées](#) que vous fournissez, l'application lance une instance EC2, puis surveille l'instance jusqu'à ce qu'elle sorte de l'état « En attente ».

Lorsque votre instance EC2 est en cours d'exécution, vous pouvez vous y connecter à distance, comme décrit dans [\(facultatif\) Connectez-vous à l'instance](#).

⚠ Warning

EC2-Classic a été retiré le 15 août 2022. Nous vous recommandons de migrer d'EC2-Classic vers un VPC. Pour plus d'informations, consultez le billet de blog [EC2-Classic Networking is Retiring — Here's How to Prepare](#).

Les sections suivantes fournissent des extraits et d'autres informations pour cet exemple. Le [code complet de l'exemple](#) est affiché après les extraits et peut être créé et exécuté tel quel.

Rubriques

- [Rassemblez ce dont vous avez besoin](#)
- [Lancer une instance](#)
- [Surveiller l'instance](#)
- [Code complet](#)
- [Considérations supplémentaires](#)
- [\(facultatif\) Connectez-vous à l'instance](#)
- [Nettoyage](#)

Rassemblez ce dont vous avez besoin

Pour lancer une instance EC2, vous aurez besoin de plusieurs éléments.

- Un [VPC](#) dans lequel l'instance sera lancée. S'il s'agit d'une instance Windows et que vous vous y connectez via RDP, le VPC aura probablement besoin d'une passerelle Internet attachée, ainsi que d'une entrée pour la passerelle Internet dans la table de routage. Pour plus d'informations, consultez [Passerelles Internet](#) dans le Guide de l'utilisateur Amazon VPC.
- L'ID d'un sous-réseau existant dans le VPC où l'instance sera lancée. Pour le trouver ou le créer facilement, vous pouvez vous connecter à la [console Amazon VPC](#), mais vous pouvez également l'obtenir par programmation en utilisant les méthodes et [CreateSubnetAsyncDescribeSubnetsAsync](#)

Note

Si vous ne fournissez pas ce paramètre, la nouvelle instance est lancée dans le VPC par défaut de votre compte.

- L'ID d'un groupe de sécurité existant appartenant au VPC où l'instance sera lancée. Pour de plus amples informations, veuillez consulter [Travailler avec des groupes de sécurité dans Amazon EC2](#).
- Si vous souhaitez vous connecter à la nouvelle instance, le groupe de sécurité mentionné précédemment doit disposer d'une règle entrante appropriée autorisant le trafic SSH sur le port 22 (instance Linux) ou le trafic RDP sur le port 3389 (instance Windows). Pour plus d'informations sur la procédure à suivre [Mise à jour des groupes de sécurité](#), voir [Considérations supplémentaires](#) notamment la fin de cette rubrique.
- L'Amazon Machine Image (AMI) qui sera utilisée pour créer l'instance. Pour plus d'informations AMIs, consultez [Amazon Machine Images \(AMIs\)](#) dans le guide de l'[utilisateur Amazon EC2](#). Consultez notamment les sections [Rechercher une AMI](#) et [Partager AMIs](#).
- Le nom d'une paire de clés EC2 existante, qui est utilisée pour se connecter à la nouvelle instance. Pour de plus amples informations, veuillez consulter [Utilisation des paires de clés Amazon EC2](#).
- Nom du fichier PEM contenant la clé privée de la paire de clés EC2 mentionnée précédemment. Le fichier PEM est utilisé lorsque vous vous [connectez à distance à](#) l'instance.

Lancer une instance

L'extrait de code suivant lance une instance EC2.

L'exemple [situé à la fin de cette rubrique](#) montre cet extrait de code en cours d'utilisation.

```
//  
// Method to launch the instances  
// Returns a list with the launched instance IDs  
private static async Task<List<string>> LaunchInstances(  
    IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
```

```

{
    var instanceIds = new List<string>();
    RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

    Console.WriteLine("\nNew instances have been created.");
    foreach (Instance item in responseLaunch.Reservation.Instances)
    {
        instanceIds.Add(item.InstanceId);
        Console.WriteLine($"  New instance: {item.InstanceId}");
    }

    return instanceIds;
}

```

Surveiller l'instance

L'extrait de code suivant surveille l'instance jusqu'à ce qu'elle sorte de l'état « En attente ».

L'exemple [situé à la fin de cette rubrique](#) montre cet extrait de code en cours d'utilisation.

Consultez la [InstanceState](#) classe pour connaître les valeurs valides de la `Instance.State.Code` propriété.

```

//
// Method to wait until the instances are running (or at least not pending)
private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
{
    Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

    int numberRunning;
    DescribeInstancesResponse responseDescribe;
    var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

    // Check every couple of seconds
    int wait = 2000;
    while(true)
    {
        // Get and check the status for each of the instances to see if it's past
        pending.
    }
}

```

```

// Once all instances are past pending, break out.
// (For this example, we are assuming that there is only one reservation.)
Console.WriteLine(".");
numberRunning = 0;
responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
foreach (Instance i in responseDescribe.Reservations[0].Instances)
{
    // Check the lower byte of State.Code property
    // Code == 0 is the pending state
    if((i.State.Code & 255) > 0) numberRunning++;
}
if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
    break;

// Wait a bit and try again (unless the user wants to stop waiting)
Thread.Sleep(wait);
if(Console.KeyAvailable)
    break;
}

Console.WriteLine("\nNo more instances are pending.");
foreach (Instance i in responseDescribe.Reservations[0].Instances)
{
    Console.WriteLine($"For {i.InstanceId}:");
    Console.WriteLine($"  VPC ID: {i.VpcId}");
    Console.WriteLine($"  Instance state: {i.State.Name}");
    Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
    Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
    Console.WriteLine($"  Key pair name: {i.KeyName}");
}
}

```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

Classe [InstanceType](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [DescribeInstancesRequest](#)

Classe [DescribeInstancesResponse](#)

[Instance](#) de classe

Classe [InstanceNetworkInterfaceSpecification](#)

Classe [RunInstancesRequest](#)

Classe [RunInstancesResponse](#)

Le code

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
    // = = = = =
    // Class to launch an EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
            }
        }
    }
}
```

```

        return;
    }

    // Get the application arguments from the parsed list
    string groupID =
        CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
    string ami =
        CommandLine.GetArgument(parsedArgs, null, "-a", "--ami-id");
    string keyPairName =
        CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
    string subnetID =
        CommandLine.GetArgument(parsedArgs, null, "-s", "--subnet-id");
    if( (string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
        || (string.IsNullOrEmpty(ami) || !ami.StartsWith("ami-"))
        || (string.IsNullOrEmpty(keyPairName))
        || (!string.IsNullOrEmpty(subnetID) && !subnetID.StartsWith("subnet-")))
        CommandLine.ErrorExit(
            "\nOne or more of the required arguments is missing or incorrect." +
            "\nRun the command with no arguments to see help.");

    // Create an EC2 client
    var ec2Client = new AmazonEC2Client();

    // Create an object with the necessary properties
    RunInstancesRequest request = GetRequestData(groupID, ami, keyPairName,
subnetID);

    // Launch the instances and wait for them to start running
    var instanceIds = await LaunchInstances(ec2Client, request);
    await CheckState(ec2Client, instanceIds);
}

//
// Method to put together the properties needed to launch the instance.
private static RunInstancesRequest GetRequestData(
    string groupID, string ami, string keyPairName, string subnetID)
{
    // Common properties
    var groupIDs = new List<string>() { groupID };
    var request = new RunInstancesRequest()
    {
        // The first three of these would be additional command-line arguments or
        similar.
    }

```

```

        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
    };

    // Properties specifically for EC2 in a VPC.
    if(!string.IsNullOrEmpty(subnetID))
    {
        request.NetworkInterfaces =
            new List<InstanceNetworkInterfaceSpecification>() {
                new InstanceNetworkInterfaceSpecification() {
                    DeviceIndex = 0,
                    SubnetId = subnetID,
                    Groups = groupIDs,
                    AssociatePublicIpAddress = true
                }
            };
    }

    // Properties specifically for EC2-Classic
    else
    {
        request.SecurityGroupIds = groupIDs;
    }
    return request;
}

//
// Method to launch the instances
// Returns a list with the launched instance IDs
private static async Task<List<string>> LaunchInstances(
    IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
{
    var instanceIds = new List<string>();
    RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

    Console.WriteLine("\nNew instances have been created.");
    foreach (Instance item in responseLaunch.Reservation.Instances)
    {
        instanceIds.Add(item.InstanceId);
    }
}

```

```

        Console.WriteLine($"  New instance: {item.InstanceId}");
    }

    return instanceIds;
}

//
// Method to wait until the instances are running (or at least not pending)
private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
{
    Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

    int numberRunning;
    DescribeInstancesResponse responseDescribe;
    var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

    // Check every couple of seconds
    int wait = 2000;
    while(true)
    {
        // Get and check the status for each of the instances to see if it's past
        pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach (Instance i in responseDescribe.Reservations[0].Instances)
        {
            // Check the lower byte of State.Code property
            // Code == 0 is the pending state
            if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
            break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)

```



```

        break;
    }

    Console.WriteLine("\nNo more instances are pending.");
    foreach (Instance i in responseDescribe.Reservations[0].Instances)
    {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
    }
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s  

<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
}

// =====
//
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //

```

```

// Returns:
// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.

```

```
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
```

Considérations supplémentaires

- Lorsque vous vérifiez l'état d'une instance EC2, vous pouvez ajouter un filtre à la `Filter` propriété de l'[DescribeInstancesRequest](#) objet. Grâce à cette technique, vous pouvez limiter la demande à certaines instances, par exemple les instances dotées d'une balise spécifique spécifiée par l'utilisateur.
- Par souci de concision, des valeurs typiques ont été attribuées à certaines propriétés. L'une ou l'ensemble de ces propriétés peuvent plutôt être déterminées par programmation ou par saisie par l'utilisateur.
- Les valeurs que vous pouvez utiliser pour les `MaxCount` propriétés `MinCount` et de l'[RunInstancesRequest](#) objet sont déterminées par la zone de disponibilité cible et le nombre maximum d'instances autorisées pour le type d'instance. Pour plus d'informations, consultez la section [Combien d'instances puis-je exécuter dans Amazon EC2](#) dans la FAQ générale d'Amazon EC2.

- Si vous souhaitez utiliser un type d'instance différent de celui de cet exemple, vous avez le choix entre plusieurs types d'instance. Pour plus d'informations, consultez les [types d'instances Amazon EC2](#) dans le guide de l'utilisateur [Amazon EC2](#). Consultez également les [détails du type d'instance et l'explorateur de types d'instance](#).
- Vous pouvez également associer un [rôle IAM](#) à une instance lorsque vous la lancez. Pour ce faire, créez un [IamInstanceProfileSpecification](#) objet dont la Name propriété est définie sur le nom d'un rôle IAM. Ajoutez ensuite cet objet à la `IamInstanceProfile` propriété de l'[RunInstancesRequest](#) objet.

 Note

Pour lancer une instance EC2 à laquelle un rôle IAM est attaché, la configuration d'un utilisateur IAM doit inclure certaines autorisations. Pour plus d'informations sur les autorisations requises, consultez la section [Accorder à un utilisateur l'autorisation de transmettre un rôle IAM à une instance](#) dans le guide de l'utilisateur [Amazon EC2](#).

(facultatif) Connectez-vous à l'instance

Une fois qu'une instance est en cours d'exécution, vous pouvez vous y connecter à distance à l'aide du client distant approprié. Pour les instances Linux et Windows, vous avez besoin de l'adresse IP publique ou du nom DNS public de l'instance. Vous avez également besoin des éléments suivants.

Pour les instances Linux

Vous pouvez utiliser un client SSH pour vous connecter à votre instance Linux. Assurez-vous que le groupe de sécurité que vous avez utilisé lorsque vous avez lancé l'instance autorise le trafic SSH sur le port 22, comme décrit dans [Mise à jour des groupes de sécurité](#).

Vous avez également besoin de la partie privée de la paire de clés que vous avez utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations, consultez [Connect to your Linux instance](#) dans le guide de l'utilisateur Amazon EC2.

Pour les instances Windows

Vous pouvez utiliser un client RDP pour vous connecter à votre instance. Assurez-vous que le groupe de sécurité que vous avez utilisé lorsque vous avez lancé l'instance autorise le trafic RDP sur le port 3389, comme décrit dans. [Mise à jour des groupes de sécurité](#)

Vous avez également besoin du mot de passe administrateur. Vous pouvez l'obtenir en utilisant l'exemple de code suivant, qui nécessite l'ID de l'instance et la partie privée de la paire de clés utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations, consultez [Connect to your Windows instance](#) dans le guide de l'utilisateur Amazon EC2.

 Warning

Cet exemple de code renvoie le mot de passe administrateur en texte brut pour votre instance.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [GetPasswordDataRequest](#)

Classe [GetPasswordDataResponse](#)

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
```

```

using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
    // = = = = =
    // Class to get the Administrator password of a Windows EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }

            // Get the application arguments from the parsed list
            string instanceID =
                CommandLine.GetArgument(parsedArgs, null, "-i", "--instance-id");
            string pemFileName =
                CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
            if( (string.IsNullOrEmpty(instanceID) || !instanceID.StartsWith("i-"))
                || (string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem")))
                CommandLine.ErrorExit(
                    "\nOne or more of the required arguments is missing or incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            // Get and display the password
            string password = await GetPassword(ec2Client, instanceID, pemFileName);
            Console.WriteLine($"Password: {password}");
        }

        //
        // Method to get the administrator password of a Windows EC2 instance
        private static async Task<string> GetPassword(
            IAmazonEC2 ec2Client, string instanceID, string pemFilename)

```

```

{
    string password = string.Empty;
    GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
            InstanceId = instanceID});
    if(response.PasswordData != null)
    {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
    }
    else
    {
        Console.WriteLine($"\\n\\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"\\n\\nIf this is a Windows instance, the password might not be
ready.");
    }
    return password;
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\\n\\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\\n  -i, --instance-id: The name of the EC2 instance." +
        "\\n  -p, --pem-filename: The name of the PEM file with the private key.");
}

// = = = = =
// = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //

```

```

// Returns:
// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.

```



```
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}

}
```

Nettoyage

Lorsque vous n'avez plus besoin de votre instance EC2, veillez à la mettre hors service, comme décrit dans [Interruption d'une instance Amazon EC2](#).

Interruption d'une instance Amazon EC2

Lorsque vous n'avez plus besoin d'une ou de plusieurs de vos instances Amazon EC2, vous pouvez les résilier.

Cet exemple vous montre comment utiliser le pour mettre fin AWS SDK pour .NET à des instances EC2. Il prend un ID d'instance en entrée.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [TerminateInstancesRequest](#)

Classe [TerminateInstancesResponse](#)

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2TerminateInstance
{
    class Program
    {
        static async Task Main(string[] args)
        {
            if((args.Length == 1) && (args[0].StartsWith("i-")))
            {
                // Terminate the instance
                var ec2Client = new AmazonEC2Client();
                await TerminateInstance(ec2Client, args[0]);
            }
            else
            {
                Console.WriteLine("\nCommand-line argument missing or incorrect.");
                Console.WriteLine("\nUsage: EC2TerminateInstance instance-ID");
                Console.WriteLine(" instance-ID - The EC2 instance you want to terminate.");
                return;
            }
        }
    }

    //
    // Method to terminate an EC2 instance
    private static async Task TerminateInstance(IAmazonEC2 ec2Client, string
instanceID)
    {
        var request = new TerminateInstancesRequest{
```

```
        InstanceIds = new List<string>() { instanceID } };
    TerminateInstancesResponse response =
        await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>() { instanceID }
        });
    foreach (InstanceStateChange item in response.TerminatingInstances)
    {
        Console.WriteLine("Terminated instance: " + item.InstanceId);
        Console.WriteLine("Instance state: " + item.CurrentState.Name);
    }
}
}
```

Après avoir exécuté l'exemple, il est conseillé de vous connecter à la [console Amazon EC2](#) pour vérifier que l'[instance EC2](#) a été résiliée.

Tutoriel sur les instances Spot Amazon EC2

Ce didacticiel explique comment utiliser le AWS SDK pour .NET pour gérer les instances Spot Amazon EC2.

Présentation de

Les instances Spot vous permettent de demander de la capacité Amazon EC2 inutilisée à un prix inférieur au prix à la demande. Cela peut réduire considérablement vos coûts EC2 pour les applications susceptibles d'être interrompues.

Vous trouverez ci-dessous un résumé détaillé de la manière dont les instances Spot sont demandées et utilisées.

1. Créez une demande d'instance Spot en spécifiant le prix maximum que vous êtes prêt à payer.
2. Lorsque la demande est satisfaite, exécutez l'instance comme n'importe quelle autre instance Amazon EC2.
3. Exécutez l'instance aussi longtemps que vous le souhaitez, puis mettez-la hors service, sauf si le prix au comptant change de telle sorte que l'instance soit résiliée pour vous.
4. Nettoyez la demande d'instance Spot lorsque vous n'en avez plus besoin afin que les instances Spot ne soient plus créées.

Il s'agit d'un aperçu de très haut niveau des instances Spot. Pour mieux comprendre les instances Spot, consultez la section [Instances Spot](#) dans le guide de l'[utilisateur Amazon EC2](#).

À propos de ce didacticiel

Au fur et à mesure que vous suivez ce didacticiel, vous utilisez le AWS SDK pour .NET pour effectuer les opérations suivantes :

- Créer une demande d'instance Spot
- Déterminez quand la demande d'instance Spot a été satisfaite
- Annuler la demande d'instance Spot
- Résilier les instances associées

Les sections suivantes fournissent des extraits et d'autres informations pour cet exemple. Le [code complet de l'exemple](#) est affiché après les extraits et peut être créé et exécuté tel quel.

Rubriques

- [Conditions préalables](#)
- [Rassemblez ce dont vous avez besoin](#)
- [Création d'une demande d'instance Spot](#)
- [Déterminez l'état de votre demande d'instance Spot](#)
- [Nettoyez vos demandes d'instance Spot](#)
- [Nettoyez vos instances Spot](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Conditions préalables

Pour plus d'informations sur les APIs et les prérequis, consultez la section parent ([Utilisation avec Amazon EC2](#)).

Rassemblez ce dont vous avez besoin

Pour créer une demande d'instance Spot, vous aurez besoin de plusieurs éléments.

- Le nombre d'instances et leur type d'instance. Vous avez le choix entre plusieurs types d'instances. Pour plus d'informations, consultez les [types d'instances Amazon EC2](#) dans le guide de l'utilisateur

[Amazon EC2](#). Consultez également les [détails du type d'instance](#) et l'[explorateur de types d'instance](#).

Le nombre par défaut pour ce didacticiel est 1.

- L'Amazon Machine Image (AMI) qui sera utilisée pour créer l'instance. Pour plus d'informations AMIs, consultez [Amazon Machine Images \(AMIs\)](#) dans le guide de l'[utilisateur Amazon EC2](#). Consultez notamment les sections [Rechercher une AMI](#) et [Partager AMIs](#).
- Le prix maximum que vous êtes prêt à payer par heure d'instance. Vous pouvez consulter les prix de tous les types d'instances (pour les instances à la demande et les instances ponctuelles) sur la page de [tarification d'Amazon EC2](#). Le prix par défaut de ce didacticiel est expliqué plus loin.
- Si vous souhaitez vous connecter à distance à une instance, un groupe de sécurité doté de la configuration et des ressources appropriées. Ceci est décrit dans [Travailler avec des groupes de sécurité dans Amazon EC2](#) les informations relatives à la [collecte de ce dont vous avez besoin](#) et [à la connexion à une instance](#) dans [Lancement d'une instance Amazon EC2](#). Pour des raisons de simplicité, ce didacticiel utilise le groupe de sécurité nommé default que possèdent tous les nouveaux AWS comptes.

Il existe de nombreuses façons d'aborder la demande d'instances Spot. Les stratégies les plus courantes sont les suivantes :

- Faites des demandes qui coûteront certainement moins cher que la tarification à la demande.
- Effectuez des demandes en fonction de la valeur du calcul obtenu.
- Faites des demandes afin d'acquérir de la capacité de calcul le plus rapidement possible.

Les explications suivantes font référence à l'[historique des tarifs des instances Spot](#) dans le guide de l'[utilisateur Amazon EC2](#).

Réduisez les coûts en dessous de la demande

Vous avez une tâche de traitement par lot dont l'exécution prendra un certain nombre d'heures ou de jours. Toutefois, vous êtes flexible quant aux dates et heures de début et de fin de la tâche. Vous voulez savoir si vous pouvez exécuter cette tâche à un coût inférieur à celui des instances à la demande.

Vous examinez l'historique des prix au comptant pour les types d'instances à l'aide de la console Amazon EC2 ou de l'API Amazon EC2. Une fois que vous avez analysé l'historique des prix pour le type d'instance souhaité dans une zone de disponibilité donnée, deux approches sont possibles pour votre demande :

- Spécifiez une demande se situant dans la partie supérieure de la fourchette des prix ponctuels, qui sont toujours inférieurs au prix à la demande, en anticipant que votre demande d'instance ponctuelle unique sera probablement satisfaite et exécutée pendant une durée de calcul consécutive suffisante pour terminer le travail.
- Spécifiez une demande à la limite inférieure de la plage de prix et envisagez de combiner plusieurs instances lancées au fil du temps via une demande persistante. Les instances vont s'exécuter suffisamment longtemps, au total, pour terminer la tâche avec un coût total plus faible.

Ne payez pas plus que la valeur du résultat

Vous avez une tâche de traitement de données à exécuter. Vous comprenez suffisamment la valeur des résultats du travail pour savoir combien ils valent en termes de coûts informatiques.

Après avoir analysé l'historique des prix au comptant pour votre type d'instance, vous choisissez un prix dont le coût du temps de calcul ne dépasse pas la valeur des résultats de la tâche. Vous créez une demande persistante et faites en sorte qu'elle s'exécute de façon intermittente selon que le prix Spot est égal ou inférieur à votre demande.

Acquérez rapidement de la capacité informatique

Vous avez un besoin imprévu à court terme de capacité supplémentaire qui n'est pas disponible via les instances à la demande. Après avoir analysé l'historique des prix au comptant pour votre type d'instance, vous choisissez un prix supérieur au prix historique le plus élevé afin d'améliorer considérablement les chances que votre demande soit traitée rapidement et vous poursuivez le calcul jusqu'à ce qu'elle soit complète.

Une fois que vous avez rassemblé ce dont vous avez besoin et que vous avez choisi une stratégie, vous êtes prêt à demander une instance Spot. Pour ce didacticiel, le prix maximal d'instance spot par défaut est défini pour être le même que celui à la demande (qui est de 0,003 \$ pour ce didacticiel). Fixer le prix de cette manière maximise les chances de satisfaire la demande.

Création d'une demande d'instance Spot

L'extrait suivant explique comment créer une demande d'instance Spot avec les éléments que vous avez collectés précédemment.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to create a Spot Instance request  
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(  
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,  
    InstanceType instanceType, string spotPrice, int instanceCount)  
{  
    var launchSpecification = new LaunchSpecification(  
        ImageId = amiId,  
        InstanceType = instanceType  
    );  
    launchSpecification.SecurityGroups.Add(securityGroupName);  
    var request = new RequestSpotInstancesRequest(  
        SpotPrice = spotPrice,  
        InstanceCount = instanceCount,  
        LaunchSpecification = launchSpecification  
    );  
  
    RequestSpotInstancesResponse result =  
        await ec2Client.RequestSpotInstancesAsync(request);  
    return result.SpotInstanceRequests[0];  
}
```

La valeur importante renvoyée par cette méthode est l'ID de demande d'instance Spot, qui est contenu dans le `SpotInstanceRequestId` membre de l'[SpotInstanceRequest](#) objet renvoyé.

Note

Toutes les instances Spot lancées vous seront facturées. Pour éviter des coûts inutiles, veuillez à [annuler toutes les demandes](#) et à [mettre fin à toutes les instances](#).

Déterminez l'état de votre demande d'instance Spot

L'extrait suivant vous montre comment obtenir des informations sur votre demande d'instance Spot. Vous pouvez utiliser ces informations pour prendre certaines décisions dans votre code, par exemple si vous souhaitez continuer à attendre qu'une demande d'instance Spot soit traitée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
```

```
// Method to get information about a Spot Instance request, including the status,  
// instance ID, etc.  
// It gets the information for a specific request (as opposed to all requests).  
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(  
    IAmazonEC2 ec2Client, string requestId)  
{  
    var describeRequest = new DescribeSpotInstanceRequestsRequest();  
    describeRequest.SpotInstanceRequestIds.Add(requestId);  
  
    DescribeSpotInstanceRequestsResponse describeResponse =  
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);  
    return describeResponse.SpotInstanceRequests[0];  
}
```

La méthode renvoie des informations sur la demande d'instance Spot, telles que l'ID de l'instance, son état et son code d'état. Pour plus d'informations sur les codes de statut des demandes d'instance Spot, consultez la section [État des demandes Spot](#) dans le guide de [l'utilisateur Amazon EC2](#).

Nettoyez vos demandes d'instance Spot

Lorsque vous n'avez plus besoin de demander des instances ponctuelles, il est important d'annuler toutes les demandes en suspens afin d'éviter qu'elles ne soient traitées à nouveau. L'extrait suivant explique comment annuler une demande d'instance Spot.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to cancel a Spot Instance request  
private static async Task CancelSpotInstanceRequest(  
    IAmazonEC2 ec2Client, string requestId)  
{  
    var cancelRequest = new CancelSpotInstanceRequestsRequest();  
    cancelRequest.SpotInstanceRequestIds.Add(requestId);  
  
    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);  
}
```

Nettoyez vos instances Spot

Pour éviter des coûts inutiles, il est important de résilier toutes les instances créées à partir de demandes d'instances Spot ; le simple fait d'annuler les demandes d'instance Spot ne mettra pas fin à vos instances, ce qui signifie qu'elles continueront à vous être facturées. L'extrait suivant explique

comment mettre fin à une instance après avoir obtenu l'identifiant d'instance d'une instance Spot active.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to terminate a Spot Instance
private static async Task TerminateSpotInstance(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    // Retrieve the Spot Instance request to check for running instances.
    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

    // If there are any running instances, terminate them
    if( (describeResponse.SpotInstanceRequests[0].Status.Code
        == "request-canceled-and-instance-running")
        || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
    {
        TerminateInstancesResponse response =
            await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
                InstanceIds = new List<string>(){
                    describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
            Console.WriteLine($"\\n Terminated instance: {item.InstanceId}");
            Console.WriteLine($" Instance state: {item.CurrentState.Name}\\n");
        }
    }
}
```

Code complet

L'exemple de code suivant appelle les méthodes décrites précédemment pour créer et annuler une demande d'instance Spot et mettre fin à une instance Spot.

Références du SDK

NuGet colis :

- [AWSSDK.EC2](#)

Éléments de programmation :

- [Espace de noms Amazon.ec2](#)

Classe [Amazon EC2 Client](#)

Classe [InstanceType](#)

- [Espace de noms Amazon.ec2.model](#)

Classe [CancelSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsResponse](#)

Classe [InstanceStateChange](#)

Classe [LaunchSpecification](#)

Classe [RequestSpotInstancesRequest](#)

Classe [RequestSpotInstancesResponse](#)

Classe [SpotInstanceRequest](#)

Classe [TerminateInstancesRequest](#)

Classe [TerminateInstancesResponse](#)

Le code

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2SpotInstanceRequests
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Some default values.
            // These could be made into command-line arguments instead.
            var instanceType = InstanceType.T1Micro;
            string securityGroupName = "default";
            string spotPrice = "0.003";
            int instanceCount = 1;

            // Parse the command line arguments
            if((args.Length != 1) || (!args[0].StartsWith("ami-")))
            {
                Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
                Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
Instances.");
                return;
            }

            // Create the Amazon EC2 client.
            var ec2Client = new AmazonEC2Client();

            // Create the Spot Instance request and record its ID
            Console.WriteLine("\nCreating spot instance request...");
            var req = await CreateSpotInstanceRequest(
                ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
            string requestId = req.SpotInstanceRequestId;

            // Wait for an EC2 Spot Instance to become active
            Console.WriteLine(
                $"Waiting for Spot Instance request with ID {requestId} to become active...");
            int wait = 1;
            var start = DateTime.Now;
            while(true)
            {
                Console.Write(".");

                // Get and check the status to see if the request has been fulfilled.
                var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
                if(requestInfo.Status.Code == "fulfilled")
                {
```

```

        Console.WriteLine($"Spot Instance request {requestId} " +
            $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
        break;
    }

    // Wait a bit and try again, longer each time (1, 2, 4, ...)
    Thread.Sleep(wait);
    wait = wait * 2;
}

// Show the user how long it took to fulfill the Spot Instance request.
TimeSpan span = DateTime.Now.Subtract(start);
Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

// Perform actions here as needed.
// For this example, simply wait for the user to hit a key.
// That gives them a chance to look at the EC2 console to see
// the running instance if they want to.
Console.WriteLine("Press any key to start the cleanup...");
Console.ReadKey(true);

// Cancel the request.
// Do this first to make sure that the request can't be re-fulfilled
// once the Spot Instance has been terminated.
Console.WriteLine("Canceling Spot Instance request...");
await CancelSpotInstanceRequest(ec2Client, requestId);

// Terminate the Spot Instance that's running.
Console.WriteLine("Terminating the running Spot Instance...");
await TerminateSpotInstance(ec2Client, requestId);

Console.WriteLine("Done. Press any key to exit...");
Console.ReadKey(true);
}

//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,
    InstanceType instanceType, string spotPrice, int instanceCount)
{
    var launchSpecification = new LaunchSpecification{
        ImageId = amiId,

```

```

        InstanceType = instanceType
    };
    launchSpecification.SecurityGroups.Add(securityGroupName);
    var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
    };

    RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
    return result.SpotInstanceRequests[0];
}

//
// Method to get information about a Spot Instance request, including the status,
// instance ID, etc.
// It gets the information for a specific request (as opposed to all requests).
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
    return describeResponse.SpotInstanceRequests[0];
}

//
// Method to cancel a Spot Instance request
private static async Task CancelSpotInstanceRequest(
    IAmazonEC2 ec2Client, string requestId)
{
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(requestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}

//

```

```
// Method to terminate a Spot Instance
private static async Task TerminateSpotInstance(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    // Retrieve the Spot Instance request to check for running instances.
    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

    // If there are any running instances, terminate them
    if(    (describeResponse.SpotInstanceRequests[0].Status.Code
            == "request-canceled-and-instance-running")
        || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
    {
        TerminateInstancesResponse response =
            await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
                InstanceIds = new List<string>(){
                    describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
            Console.WriteLine($"\\n  Terminated instance: {item.InstanceId}");
            Console.WriteLine($"  Instance state: {item.CurrentState.Name}\\n");
        }
    }
}
}
```

Considérations supplémentaires

- Après avoir exécuté le didacticiel, il est conseillé de vous connecter à la [console Amazon EC2](#) pour vérifier que la [demande d'instance Spot](#) a été annulée et que l'[instance Spot](#) a été résiliée.

Accès Gestion des identités et des accès AWS (IAM) à l'aide du AWS SDK pour .NET

The AWS SDK pour .NET supports [Gestion des identités et des accès AWS](#), qui est un service Web qui permet aux AWS clients de gérer les utilisateurs et les autorisations des utilisateurs dans AWS.

Un utilisateur Gestion des identités et des accès AWS (IAM) est une entité que vous créez dans AWS. L'entité représente une personne ou une application qui interagit avec AWS. Pour plus d'informations sur les utilisateurs IAM, voir Utilisateurs [IAM et limites IAM et STS dans le Guide de l'utilisateur IAM](#).

Vous accordez des autorisations à un utilisateur en créant une politique IAM. La politique contient un document de politique qui répertorie les actions qu'un utilisateur peut effectuer et les ressources que ces actions peuvent affecter. Pour plus d'informations sur les politiques IAM, consultez la section [Politiques et autorisations](#) dans le guide de l'utilisateur IAM.

Warning

Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que [AWS IAM Identity Center](#).

APIs

Les AWS SDK pour .NET offrent des APIs pour les clients IAM. Ils vous permettent de travailler avec les fonctionnalités IAM telles que les utilisateurs, les rôles et les clés d'accès.

Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la [référence de l'AWS SDK pour .NET API](#) (et faites défiler la page jusqu'à « Amazon »). IdentityManagement«).

Cette section contient également [un exemple](#) qui explique comment associer un rôle IAM aux instances Amazon EC2 afin de faciliter la gestion des informations d'identification.

[Les IAM APIs sont fournis par leAWSSDK. IdentityManagement](#) NuGetcolis.

Conditions préalables

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

Rubriques

- [Création de politiques gérées par IAM à partir de JSON](#)
- [Afficher le document de politique d'une politique gérée par IAM](#)
- [Octroi d'accès à l'aide d'un rôle IAM](#)

Création de politiques gérées par IAM à partir de JSON

Cet exemple vous montre comment utiliser le pour créer une [politique gérée par IAM AWS SDK pour .NET](#) à partir d'un document de stratégie donné au format JSON. L'application crée un objet client IAM, lit le document de politique à partir d'un fichier, puis crée la stratégie.

Note

Pour un exemple de document de politique au format JSON, consultez les [considérations supplémentaires](#) à la fin de cette rubrique.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Création de la stratégie](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Création de la stratégie

L'extrait de code suivant crée une politique gérée par IAM avec le nom et le document de stratégie indiqués.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to create an IAM policy from a JSON file  
private static async Task<CreatePolicyResponse> CreateManagedPolicy(  
    IAmazonIdentityManagementService iamClient, string policyName, string  
    jsonFilename)  
{  
    return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
```



```

        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename));
    }

```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.IdentityManagement](#)

Éléments de programmation :

- Espace de noms [Amazon. IdentityManagement](#)
 Classe [AmazonIdentityManagementServiceClient](#)
- Espace de noms [Amazon. IdentityManagement.Modèle](#)
 Classe [CreatePolicyRequest](#)
 Classe [CreatePolicyResponse](#)

Le code

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
    // = = = = =
    // Class to create an IAM policy with a given policy document
    class Program
    {
        private const int MaxArgs = 2;
    }
}

```

```

static async Task Main(string[] args)
{
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
    {
        PrintHelp();
        return;
    }

    // Get the application arguments from the parsed list
    string policyName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-name");
    string policyFilename =
        CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
    if(    string.IsNullOrEmpty(policyName)
        || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
        CommandLine.ErrorExit(
            "\nOne or more of the required arguments is missing or incorrect." +
            "\nRun the command with no arguments to see help.");

    // Create an IAM service client
    var iamClient = new AmazonIdentityManagementServiceClient();

    // Create the new policy
    var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
    Console.WriteLine($"Policy {response.Policy.PolicyName} has been created.");
    Console.WriteLine($"  Arn: {response.Policy.Arn}");
}

//
// Method to create an IAM policy from a JSON file
private static async Task<CreatePolicyResponse> CreateManagedPolicy(
    IAmazonIdentityManagementService iamClient, string policyName, string
jsonFilename)
{
    return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
}

```

```

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
document.");
}
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];

```

```

        var value = key;

        // Check to see if there's a value that goes with this option?
        if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
        parsedArgs.Add(key, value);
    }

    // If the first argument in this iteration doesn't start with a dash, it's a
value
    else
    {
        parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
        n++;
    }
}

return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}

```

```
}  
}  
  
}
```

Considérations supplémentaires

- Voici un exemple de document de politique que vous pouvez copier dans un fichier JSON et utiliser comme entrée pour cette application :

JSON

```
{  
  "Version": "2012-10-17",  
  "Id" : "DotnetTutorialPolicy",  
  "Statement" : [  
    {  
      "Sid" : "DotnetTutorialPolicyS3",  
      "Effect" : "Allow",  
      "Action" : [  
        "s3:Get*",  
        "s3:List*"  
      ],  
      "Resource" : "*"  
    },  
    {  
      "Sid" : "DotnetTutorialPolicyPolly",  
      "Effect": "Allow",  
      "Action": [  
        "polly:DescribeVoices",  
        "polly:SynthesizeSpeech"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```

- Vous pouvez vérifier que la politique a été créée en consultant la [console IAM](#). Dans la liste déroulante des politiques de filtrage, sélectionnez Géré par le client. Supprimez la politique lorsque vous n'en avez plus besoin.

- Pour plus d'informations sur la création de politiques, consultez [Création de politiques IAM](#) et la [référence de politique IAM JSON](#) dans le guide de l'utilisateur [IAM](#)

Afficher le document de politique d'une politique gérée par IAM

Cet exemple montre comment utiliser le AWS SDK pour .NET pour afficher un document de politique. L'application crée un objet client IAM, trouve la version par défaut de la politique gérée IAM donnée, puis affiche le document de stratégie au format JSON.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Trouvez la version par défaut](#)
- [Afficher le document de politique](#)
- [Code complet](#)

Trouvez la version par défaut

L'extrait suivant trouve la version par défaut de la politique IAM donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
    IAMazonIdentityManagementService iamClient, string policyArn)
{
    // Retrieve all the versions of this policy
    string defaultVersion = string.Empty;
    ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
            PolicyArn = policyArn});

    // Find the default version
    foreach(PolicyVersion version in reponseVersions.Versions)
    {
        if(version.IsDefaultVersion)
        {
```

```
        defaultVersion = version.VersionId;
        break;
    }
}

return defaultVersion;
}
```

Afficher le document de politique

L'extrait suivant affiche le document de politique au format JSON de la stratégie IAM donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to retrieve and display the policy document of an IAM policy
private static async Task ShowPolicyDocument(
    IAmazonIdentityManagementService iamClient, string policyArn, string
    defaultVersion)
{
    // Retrieve the policy document of the default version
    GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
            PolicyArn = policyArn,
            VersionId = defaultVersion});

    // Display the policy document (in JSON)
    Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
    Console.WriteLine(
        $"{{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}}");
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.IdentityManagement](#)

Éléments de programmation :

- Espace de noms [Amazon. IdentityManagement](#)

Classe [AmazonIdentityManagementServiceClient](#)

- Espace de noms [Amazon. IdentityManagement.Modèle](#)

Classe [GetPolicyVersionRequest](#)

Classe [GetPolicyVersionResponse](#)

Classe [ListPolicyVersionsRequest](#)

Classe [ListPolicyVersionsResponse](#)

Classe [PolicyVersion](#)

Le code

```
using System;
using System.Web;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamDisplayPolicyJson
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            if(args.Length != 1)
            {
                Console.WriteLine("\nUsage: IamDisplayPolicyJson policy-arn");
                Console.WriteLine("    policy-arn: The ARN of the policy to retrieve.");
                return;
            }
            if(!args[0].StartsWith("arn:"))
            {
                Console.WriteLine("\nCould not find policy ARN in the command-line arguments:");
                Console.WriteLine($"{args[0]}");
                return;
            }
        }
    }
}
```



```

    }

    // Create an IAM service client
    var iamClient = new AmazonIdentityManagementServiceClient();

    // Retrieve and display the policy document of the given policy
    string defaultVersion = await GetDefaultVersion(iamClient, args[0]);
    if(string.IsNullOrEmpty(defaultVersion))
        Console.WriteLine($"Could not find the default version for policy {args[0]}.");
    else
        await ShowPolicyDocument(iamClient, args[0], defaultVersion);
}

//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
    IAmazonIdentityManagementService iamClient, string policyArn)
{
    // Retrieve all the versions of this policy
    string defaultVersion = string.Empty;
    ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
            PolicyArn = policyArn});

    // Find the default version
    foreach(PolicyVersion version in reponseVersions.Versions)
    {
        if(version.IsDefaultVersion)
        {
            defaultVersion = version.VersionId;
            break;
        }
    }

    return defaultVersion;
}

//
// Method to retrieve and display the policy document of an IAM policy
private static async Task ShowPolicyDocument(

```

```
    IAmazonIdentityManagementService iamClient, string policyArn, string
    defaultVersion)
    {
        // Retrieve the policy document of the default version
        GetPolicyVersionResponse responsePolicy =
            await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
                PolicyArn = policyArn,
                VersionId = defaultVersion});

        // Display the policy document (in JSON)
        Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format:");
        Console.WriteLine(
            $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
}
```

Octroi d'accès à l'aide d'un rôle IAM

Ce didacticiel explique comment utiliser le AWS SDK pour .NET pour activer les rôles IAM sur les instances Amazon EC2.

Présentation de

Toutes les demandes AWS doivent être signées cryptographiquement à l'aide des informations d'identification émises par AWS. Par conséquent, vous avez besoin d'une stratégie pour gérer les informations d'identification des applications qui s'exécutent sur des instances Amazon EC2. Vous devez distribuer, stocker et faire pivoter ces informations d'identification en toute sécurité, mais également les garder accessibles aux applications.

Les rôles IAM vous permettent de gérer efficacement ces informations d'identification. Vous créez un rôle IAM et vous le configurez avec les autorisations requises par une application, puis vous attachez ce rôle à une instance EC2. Pour en savoir plus sur les avantages de l'utilisation des rôles IAM, consultez la section Rôles [IAM pour Amazon EC2 dans](#) le guide de l'utilisateur Amazon [EC2](#). Consultez également les informations sur les [rôles IAM](#) dans le guide de l'utilisateur IAM.

Pour une application créée à l'aide du AWS SDK pour .NET, lorsque l'application construit un objet client pour un AWS service, l'objet recherche des informations d'identification provenant de plusieurs sources potentielles. L'ordre dans lequel la recherche est effectué est indiqué dans [Résolution des informations d'identification et des profils](#).

Si l'objet client ne trouve pas d'informations d'identification provenant d'une autre source, il récupère des informations d'identification temporaires dotées des mêmes autorisations que celles qui ont été configurées dans le rôle IAM et figurent dans les métadonnées de l'instance EC2. Ces informations d'identification sont utilisées pour effectuer des appels AWS depuis l'objet client.

À propos de ce didacticiel

Au fur et à mesure que vous suivez ce didacticiel, vous utilisez le AWS SDK pour .NET (et d'autres outils) pour lancer une instance Amazon EC2 avec un rôle IAM attaché, puis vous verrez une application sur l'instance utilisant les autorisations du rôle IAM.

Rubriques

- [Création d'un exemple d'application Amazon S3](#)
- [Créer un rôle IAM](#)
- [Lancez une instance EC2 et attachez le rôle IAM](#)
- [Connectez-vous à l'instance EC2](#)
- [Exécutez l'exemple d'application sur l'instance EC2](#)
- [Nettoyage](#)

Création d'un exemple d'application Amazon S3

Cet exemple d'application récupère un objet depuis Amazon S3. Pour exécuter l'application, vous avez besoin des éléments suivants :

- Un compartiment Amazon S3 contenant un fichier texte.
- AWS informations d'identification sur votre machine de développement qui vous permettent d'accéder au bucket.

Pour plus d'informations sur la création d'un compartiment Amazon S3 et le téléchargement d'un objet, consultez le [guide de l'utilisateur d'Amazon Simple Storage Service](#). Pour plus d'informations sur les AWS informations d'identification, consultez [Configurer l'authentification du SDK avec AWS](#).

Créez un projet .NET Core avec le code suivant. Testez ensuite l'application sur votre machine de développement.

Note

Sur votre machine de développement, le .NET Core Runtime est installé, ce qui vous permet d'exécuter l'application sans la publier. Lorsque vous créez une instance EC2 ultérieurement dans ce didacticiel, vous pouvez choisir d'installer le .NET Core Runtime sur l'instance. Cela vous donne une expérience similaire et un transfert de fichiers plus réduit. Toutefois, vous pouvez également choisir de ne pas installer le .NET Core Runtime sur l'instance. Si vous choisissez cette option, vous devez publier l'application afin que toutes les dépendances soient incluses lorsque vous la transférez vers l'instance.

Références du SDK

NuGet colis :

- [AWSSDKS.3](#)

Éléments de programmation :

- [Espace de noms Amazon.S3](#)

Classe : [Amazon S3 Client](#)

- [Espace de noms Amazon.S3.Model](#)

Classe [GetObjectResponse](#)

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
    // = = = = =
    // Class to retrieve a text file from an S3 bucket and write it to a local file
```

```

class Program
{
    static async Task Main(string[] args)
    {
        // Parse the command line and show help if necessary
        var parsedArgs = CommandLine.Parse(args);
        if(parsedArgs.Count == 0)
        {
            PrintHelp();
            return;
        }

        // Get the application arguments from the parsed list
        string bucket =
            CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
        string item =
            CommandLine.GetArgument(parsedArgs, null, "-t", "--text-object");
        string outFile =
            CommandLine.GetArgument(parsedArgs, null, "-o", "--output-filename");
        if(    string.IsNullOrEmpty(bucket)
            || string.IsNullOrEmpty(item)
            || string.IsNullOrEmpty(outFile))
            CommandLine.ErrorExit(
                "\nOne or more of the required arguments is missing or incorrect." +
                "\nRun the command with no arguments to see help.");

        // Create the S3 client object and get the file object from the bucket.
        var response = await GetObject(new AmazonS3Client(), bucket, item);

        // Write the contents of the file object to the given output file.
        var reader = new StreamReader(response.ResponseStream);
        string contents = reader.ReadToEnd();
        using (var s = new FileStream(outFile, FileMode.Create))
        using (var writer = new StreamWriter(s))
            writer.WriteLine(contents);
    }

    //
    // Method to get an object from an S3 bucket.
    private static async Task<GetObjectResponse> GetObject(
        IAmazonS3 s3Client, string bucket, string item)
    {
        Console.WriteLine($"Retrieving {item} from bucket {bucket}.");
    }
}

```

```

        return await s3Client.GetObjectAsync(bucket, item);
    }

    //
    // Command-line help
    private static void PrintHelp()
    {
        Console.WriteLine(
            "\nUsage: S3GetTextItem -b <bucket-name> -t <text-object> -o <output-filename>"
+
            "\n  -b, --bucket-name: The name of the S3 bucket." +
            "\n  -t, --text-object: The name of the text object in the bucket." +
            "\n  -o, --output-filename: The name of the file to write the text to.");
    }
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {

```

```

        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)

```

```
{  
    Console.WriteLine("\nError");  
    Console.WriteLine(msg);  
    Environment.Exit(code);  
}  
}
```

Si vous le souhaitez, vous pouvez supprimer temporairement les informations d'identification que vous utilisez sur votre machine de développement pour voir comment l'application répond. (Mais assurez-vous de restaurer les informations d'identification lorsque vous aurez terminé.)

Créer un rôle IAM

Créez un rôle IAM doté des autorisations appropriées pour accéder à Amazon S3.

1. Ouvrez la [console IAM](#).
2. Dans le volet de navigation, choisissez Rôles, puis Create Role.
3. Sélectionnez le AWS service, recherchez et choisissez EC2, puis choisissez Next : Permissions.
4. Sous Joindre les politiques d'autorisation, recherchez et sélectionnez AmazonS3ReadOnlyAccess. Passez en revue la politique si vous le souhaitez, puis choisissez Next : Tags.
5. Ajoutez des balises si vous le souhaitez, puis choisissez Suivant : Révision.
6. Tapez un nom et une description pour le rôle, puis choisissez Créer un rôle. Conservez bien ce nom, car il vous sera utile lorsque vous lancerez votre instance EC2.

Lancez une instance EC2 et attachez le rôle IAM

Lancez une instance EC2 avec le rôle IAM que vous avez créé précédemment. Vous pouvez le faire de différentes manières.

- Utilisation de la console EC2

Pour lancer une instance à l'aide de la console EC2, consultez [Lancer une instance à l'aide du nouvel assistant de lancement d'instance](#) dans le guide de l'[utilisateur Amazon EC2](#).

Lorsque vous parcourez la page de lancement, vous devez au moins développer le volet Détails avancés afin de pouvoir spécifier le rôle IAM que vous avez créé précédemment dans le profil d'instance IAM.

- À l'aide du AWS SDK pour .NET

Pour plus d'informations à ce sujet [Lancement d'une instance Amazon EC2](#), voir, y compris la [Considérations supplémentaires](#) fin de cette rubrique.

Pour lancer une instance EC2 à laquelle un rôle IAM est attaché, la configuration d'un utilisateur IAM doit inclure certaines autorisations. Pour plus d'informations sur les autorisations requises, consultez la section [Accorder à un utilisateur l'autorisation de transmettre un rôle IAM à une instance](#) dans le guide de l'utilisateur [Amazon EC2](#).

Connectez-vous à l'instance EC2

Connectez-vous à l'instance EC2 afin de pouvoir y transférer l'exemple d'application, puis exécuter l'application. Vous aurez besoin du fichier contenant la partie privée de la paire de clés que vous avez utilisée pour lancer l'instance, c'est-à-dire le fichier PEM.

Pour plus d'informations sur la connexion à une instance, consultez [Connect to your Linux instance](#) ou [Connect to your Windows instance](#) dans le guide de l'[utilisateur Amazon EC2](#). Lorsque vous vous connectez, faites-le de manière à pouvoir transférer des fichiers de votre machine de développement vers votre instance.

Si vous utilisez Visual Studio sous Windows, vous pouvez également vous connecter à l'instance à l'aide du Toolkit for Visual Studio. Pour plus d'informations, consultez la section [Connexion à une instance Amazon EC2](#) dans le guide de l' AWS Toolkit for Visual Studio utilisateur.

Exécutez l'exemple d'application sur l'instance EC2

1. Copiez les fichiers d'application de votre disque local vers votre instance.

Les fichiers que vous transférez dépendent de la façon dont vous avez créé l'application et de l'installation du .NET Core Runtime sur votre instance. Pour plus d'informations sur le transfert de fichiers vers votre instance, consultez [Connect to your Linux](#) (voir la sous-section appropriée) ou [Transférer des fichiers vers des instances Windows](#) dans le guide de l'[utilisateur Amazon EC2](#).

2. Lancez l'application et vérifiez qu'elle s'exécute avec les mêmes résultats que sur votre machine de développement.
3. Vérifiez que l'application utilise les informations d'identification fournies par le rôle IAM.
 - a. Ouvrez la [console Amazon EC2](#).

- b. Sélectionnez l'instance et détachez le rôle IAM via Actions, Paramètres de l'instance, Attacher/remplacer le rôle IAM.
- c. Exécutez à nouveau l'application et vérifiez qu'elle renvoie une erreur d'autorisation.

Nettoyage

Lorsque vous aurez terminé ce didacticiel, et si vous ne souhaitez plus utiliser l'instance EC2 que vous avez créée, veillez à mettre fin à l'instance pour éviter des coûts inutiles. Vous pouvez le faire dans la [console Amazon EC2](#) ou par programmation, comme décrit dans [Interruption d'une instance Amazon EC2](#). Si vous le souhaitez, vous pouvez également supprimer d'autres ressources que vous avez créées pour ce didacticiel. Il peut s'agir d'un rôle IAM, d'une paire de clés EC2 et d'un fichier PEM, d'un groupe de sécurité, etc.

Utilisation du stockage sur Internet d'Amazon Simple Storage Service

Il AWS SDK pour .NET prend en charge [Amazon S3](#), qui est un stockage pour Internet. Il est conçu pour faciliter l'informatique à l'échelle d'Internet pour les développeurs.

APIs

Les AWS SDK pour .NET offrent des APIs pour les clients Amazon S3. Ils vous permettent de travailler avec des ressources Amazon S3 telles que des compartiments et des articles. Pour consulter l'ensemble complet des APIs de Amazon S3, consultez ce qui suit :

- [AWS SDK pour .NET Référence d'API](#) (et faites défiler la page jusqu'à « Amazon.S3 »).
- [Amazon.Extensions.S3. Documentation sur](#) le chiffrement

Les Amazon S3 APIs sont fournis par les NuGet packages suivants :

- [AWSSDKS.3](#)
- [Amazon.Extensions.S3. Chiffrement](#)

Conditions préalables

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Exemples présentés dans ce document

Les rubriques suivantes de ce document vous montrent comment utiliser le AWS SDK pour .NET pour travailler avec Amazon S3.

- [Utilisation de clés KMS pour le chiffrement S3](#)

Exemples dans d'autres documents

Les liens suivants vers le [guide du développeur Amazon S3](#) fournissent des exemples supplémentaires de la façon d'utiliser le AWS SDK pour .NET pour travailler avec Amazon S3.

Note

Bien que ces exemples et considérations de programmation supplémentaires aient été créés pour la AWS SDK pour .NET version 3 du .NET Framework, ils sont également viables pour les versions ultérieures AWS SDK pour .NET de .NET Core. De petits ajustements du code sont parfois nécessaires.

Exemples de programmation Amazon S3

- [Gérer ACLs](#)
- [Création d'un compartiment](#)
- [Chargement d'un objet](#)
- [Chargement en plusieurs parties avec l'API de haut niveau \(Amazon.S3.Transfer\). TransferUtility\)](#)
- [Chargement partitionné avec l'API de bas niveau](#)
- [Liste des objets](#)
- [Etablissement de la liste des clés](#)
- [Obtention d'un objet](#)
- [Copie d'un objet](#)
- [Copie d'un objet grâce à l'API de chargement partitionné](#)
- [Suppression d'un objet](#)
- [Suppression de plusieurs objets](#)
- [Restauration d'un objet](#)

- [Configuration des notifications d'un compartiment](#)
- [Gestion du cycle de vie d'un objet](#)
- [Génération d'une URL d'objet présignée](#)
- [Gestion de sites web](#)
- [Activation du partage des ressources de plusieurs origines \(CORS\)](#)

Considérations supplémentaires relatives à la programmation

- [Utilisation du AWS SDK pour .NET pour la programmation d'Amazon S3](#)
- [Demandes grâce aux informations d'identification temporaires de l'utilisateur IAM](#)
- [Demandes grâce aux informations d'identification de sécurité temporaires de l'utilisateur fédéré](#)
- [Spécification du chiffrement côté serveur](#)
- [Spécification du chiffrement côté serveur avec clés de chiffrement fournies par le client](#)

Utilisation de AWS KMS clés pour le chiffrement Amazon S3 dans le AWS SDK pour .NET

Cet exemple vous montre comment utiliser des AWS Key Management Service clés pour chiffrer des objets Amazon S3. L'application crée une clé principale client (CMK) et l'utilise pour créer un objet [AmazonS3 EncryptionClient V2](#) pour le chiffrement côté client. L'application utilise ce client pour créer un objet chiffré à partir d'un fichier texte donné dans un compartiment Amazon S3 existant. Il déchiffre ensuite l'objet et affiche son contenu.

Warning

Une classe similaire appelée `AmazonS3EncryptionClient` est obsolète et est moins sécurisée que la `AmazonS3EncryptionClientV2` classe. Pour migrer le code existant qui utilise `AmazonS3EncryptionClient`, voir [Migration du client de chiffrement S3 \(V1 vers V2\)](#).

Rubriques

- [Création de matériel de chiffrement](#)
- [Création et chiffrement d'un objet Amazon S3](#)
- [Code complet](#)

- [Considérations supplémentaires](#)

Création de matériel de chiffrement

L'extrait de code suivant crée un `EncryptionMaterials` objet contenant un ID de clé KMS.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
// Create a customer master key (CMK) and store the result
CreateKeyResponse createKeyResponse =
    await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
var kmsEncryptionContext = new Dictionary<string, string>();
var kmsEncryptionMaterials = new EncryptionMaterialsV2(
    createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);
```

Création et chiffrement d'un objet Amazon S3

L'extrait de code suivant crée un `AmazonS3EncryptionClientV2` objet qui utilise les matériaux de chiffrement créés précédemment. Il utilise ensuite le client pour créer et chiffrer un nouvel objet Amazon S3.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to create and encrypt an object in an S3 bucket
static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
    EncryptionMaterialsV2 materials, string bucketName,
    string fileName, string itemName)
{
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
    var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
    {
        StorageMode = CryptoStorageMode.ObjectMetadata
    };
    var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

    // Create, encrypt, and put the object
    await s3EncClient.PutObjectAsync(new PutObjectRequest
    {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
    });
}
```

```
});  
  
// Get, decrypt, and return the object  
return await s3EncClient.GetObjectAsync(new GetObjectRequest  
{  
    BucketName = bucketName,  
    Key = itemName  
});  
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [Amazon.Extensions.S3.Chiffrement](#)

Éléments de programmation :

- [Espace de noms Amazon.Extensions.S3.Encryption](#)

Classe [Amazon S3 V2 EncryptionClient](#)

Classe [Amazon S3 V2 CryptoConfiguration](#)

Classe [CryptoStorageMode](#)

Classe [EncryptionMaterialsV2](#)

- [Espace de noms Amazon.Extensions.S3.Encryption.Primitives](#)

Classe [KmsType](#)

- [Espace de noms Amazon.S3.Model](#)

Classe [GetObjectRequest](#)

Classe [GetObjectResponse](#)

Classe [PutObjectRequest](#)

- Espace de noms [Amazon.KeyManagementService](#)

Classe [AmazonKeyManagementServiceClient](#)

- Espace de noms [Amazon. KeyManagementService.Modèle](#)

Classe [CreateKeyRequest](#)

Classe [CreateKeyResponse](#)

Le code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace KmsS3Encryption
{
    // = = = = =
    // Class to store text in an encrypted S3 object.
    class Program
    {
        private const int MaxArgs = 3;

        public static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
            {
                PrintHelp();
                return;
            }

            // Get the application arguments from the parsed list
            string bucketName =
                CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
            string fileName =
```

```

        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
string itemName =
    CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
    CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");
if(!File.Exists(fileName))
    CommandLine.ErrorExit($"The given file {fileName} doesn't exist.");
if(string.IsNullOrEmpty(itemName))
    itemName = Path.GetFileName(fileName);

// Create a customer master key (CMK) and store the result
CreateKeyResponse createKeyResponse =
    await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
var kmsEncryptionContext = new Dictionary<string, string>();
var kmsEncryptionMaterials = new EncryptionMaterialsV2(
    createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

// Create the object in the bucket, then display the content of the object
var putObjectResponse =
    await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
Stream stream = putObjectResponse.ResponseStream;
StreamReader reader = new StreamReader(stream);
Console.WriteLine(reader.ReadToEnd());
}

//
// Method to create and encrypt an object in an S3 bucket
static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
    EncryptionMaterialsV2 materials, string bucketName,
    string fileName, string itemName)
{
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
    var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
    {
        StorageMode = CryptoStorageMode.ObjectMetadata
    };
    var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

    // Create, encrypt, and put the object

```



```

    await s3EncClient.PutObjectAsync(new PutObjectRequest
    {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
    });

    // Get, decrypt, and return the object
    return await s3EncClient.GetObjectAsync(new GetObjectRequest
    {
        BucketName = bucketName,
        Key = itemName
    });
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n          If item-name isn't given, file-name will be used.");
}

}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //

```

```

// Returns:
// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.

```

```
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
```

Considérations supplémentaires

- Vous pouvez vérifier les résultats de cet exemple. Pour ce faire, accédez à la [console Amazon S3](#) et ouvrez le compartiment que vous avez fourni à l'application. Recherchez ensuite le nouvel objet, téléchargez-le et ouvrez-le dans un éditeur de texte.
- La classe [AmazonS3 EncryptionClient V2](#) implémente la même interface que la classe standard `AmazonS3Client`. Cela facilite le portage de votre code vers la `AmazonS3EncryptionClientV2` classe afin que le chiffrement et le déchiffrement se fassent automatiquement et de manière transparente dans le client.
- L'un des avantages de l'utilisation d'une AWS KMS clé comme clé principale est que vous n'avez pas besoin de stocker et de gérer vos propres clés principales ; cela se fait par AWS. Un deuxième avantage est que la `AmazonS3EncryptionClientV2` classe du AWS SDK pour .NET est interopérable avec la `AmazonS3EncryptionClientV2` classe du AWS SDK pour Java. Cela signifie que vous pouvez chiffrer avec le AWS SDK pour Java et déchiffrer avec le AWS SDK pour .NET, et vice versa.

Note

La `AmazonS3EncryptionClientV2` classe des AWS SDK pour .NET prend en charge les clés principales KMS uniquement lorsqu'elle est exécutée en mode métadonnées. Le mode de fichier d'instructions de la `AmazonS3EncryptionClientV2` classe du AWS SDK pour .NET est incompatible avec la `AmazonS3EncryptionClientV2` classe du AWS SDK pour Java.

- Pour plus d'informations sur le chiffrement côté client avec cette `AmazonS3EncryptionClientV2` classe et sur le fonctionnement du chiffrement d'enveloppe, consultez la section [Chiffrement des données côté client avec Amazon AWS SDK pour .NET S3](#).

Envoyer des notifications depuis le cloud à l'aide d'Amazon Simple Notification Service

Note

Les informations contenues dans cette rubrique sont spécifiques aux projets basés sur .NET Framework et les AWS SDK pour .NET versions 3.3 et antérieures.

Il AWS SDK pour .NET prend en charge Amazon Simple Notification Service (Amazon SNS), un service Web qui permet aux applications, aux utilisateurs finaux et aux appareils d'envoyer instantanément des notifications depuis le cloud. Pour plus d'informations, consultez [Amazon SNS](#).

Répertorier vos rubriques Amazon SNS

L'exemple suivant montre comment répertorier vos rubriques Amazon SNS, les abonnements à chaque rubrique et les attributs de chaque rubrique. Cet exemple utilise la valeur par défaut [AmazonSimpleNotificationServiceClient](#).

```
// using Amazon.SimpleNotificationService;  
// using Amazon.SimpleNotificationService.Model;  
  
var client = new AmazonSimpleNotificationServiceClient();  
var request = new ListTopicsRequest();
```

```
var response = new ListTopicsResponse();

do
{
    response = client.ListTopics(request);

    foreach (var topic in response.Topics)
    {
        Console.WriteLine("Topic: {0}", topic.TopicArn);

        var subs = client.ListSubscriptionsByTopic(
            new ListSubscriptionsByTopicRequest
            {
                TopicArn = topic.TopicArn
            });

        var ss = subs.Subscriptions;

        if (ss.Any())
        {
            Console.WriteLine("  Subscriptions:");

            foreach (var sub in ss)
            {
                Console.WriteLine("    {0}", sub.SubscriptionArn);
            }
        }

        var attrs = client.GetTopicAttributes(
            new GetTopicAttributesRequest
            {
                TopicArn = topic.TopicArn
            }).Attributes;

        if (attrs.Any())
        {
            Console.WriteLine("  Attributes:");

            foreach (var attr in attrs)
            {
                Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
            }
        }
    }
}
```

```
    Console.WriteLine();  
}  
  
    request.NextToken = response.NextToken;  
  
} while (!string.IsNullOrEmpty(response.NextToken));
```

Envoyer un message à une rubrique Amazon SNS

L'exemple suivant montre comment envoyer un message à une rubrique Amazon SNS. L'exemple prend un argument, l'ARN de la rubrique Amazon SNS.

```
using System;  
using System.Linq;  
using System.Threading.Tasks;  
  
using Amazon;  
using Amazon.SimpleNotificationService;  
using Amazon.SimpleNotificationService.Model;  
  
namespace SnsSendMessage  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            /* Topic ARNs must be in the correct format:  
             *   arn:aws:sns:REGION:ACCOUNT_ID:NAME  
             *  
             *   where:  
             *   REGION      is the region in which the topic is created, such as us-  
west-2  
             *   ACCOUNT_ID is your (typically) 12-character account ID  
             *   NAME        is the name of the topic  
             */  
            string topicArn = args[0];  
            string message = "Hello at " + DateTime.Now.ToShortTimeString();  
  
            var client = new AmazonSimpleNotificationServiceClient(region:  
Amazon.RegionEndpoint.USSouth2);  
  
            var request = new PublishRequest  
            {
```

```

        Message = message,
        TopicArn = topicArn
    };

    try
    {
        var response = client.Publish(request);

        Console.WriteLine("Message sent to topic:");
        Console.WriteLine(message);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Caught exception publishing request:");
        Console.WriteLine(ex.Message);
    }
}
}
}

```

Consultez l'[exemple complet](#), y compris les informations sur la façon de créer et d'exécuter l'exemple à partir de la ligne de commande, sur GitHub.

Envoi d'un SMS à un numéro de téléphone

L'exemple suivant montre comment envoyer un SMS à un numéro de téléphone. L'exemple utilise un argument, le numéro de téléphone, qui doit être dans l'un des deux formats décrits dans les commentaires.

```

using System;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsPublish
{
    class Program
    {
        static void Main(string[] args)
        {
            // US phone numbers must be in the correct format:

```

```

        // +1 (nnn) nnn-nnnn OR +1nnnnnnnnnn
        string number = args[0];
        string message = "Hello at " + DateTime.Now.ToShortTimeString();

        var client = new AmazonSimpleNotificationServiceClient(region:
Amazon.RegionEndpoint.USSouth2);
        var request = new PublishRequest
        {
            Message = message,
            PhoneNumber = number
        };

        try
        {
            var response = client.Publish(request);

            Console.WriteLine("Message sent to " + number + ":");
            Console.WriteLine(message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Caught exception publishing request:");
            Console.WriteLine(ex.Message);
        }
    }
}

```

Consultez l'[exemple complet](#), y compris les informations sur la façon de créer et d'exécuter l'exemple à partir de la ligne de commande, sur GitHub.

Messagerie à l'aide d'Amazon SQS

Il AWS SDK pour .NET prend en charge [Amazon Simple Queue Service \(Amazon SQS\)](#), un service de mise en file d'attente de messages qui gère les messages ou les flux de travail entre les composants d'un système.

Les files d'attente Amazon SQS fournissent un mécanisme qui vous permet d'envoyer, de stocker et de recevoir des messages entre des composants logiciels tels que des microservices, des systèmes distribués et des applications sans serveur. Cela vous permet de dissocier ces composants et de vous affranchir de la nécessité de concevoir et d'exploiter votre propre système de messagerie. Pour plus d'informations sur le fonctionnement des files d'attente et des messages dans Amazon SQS,

consultez les didacticiels [Amazon SQS](#) et l'[architecture de base d'Amazon SQS](#) dans le manuel [Amazon Simple Queue Service Developer Guide](#).

Important

En raison de la nature distribuée des files d'attente, Amazon SQS ne peut garantir que vous recevrez les messages dans l'ordre précis dans lequel ils sont envoyés. Si vous devez préserver l'ordre des messages, utilisez une file d'attente [FIFO Amazon SQS](#).

APIs

Les AWS SDK pour .NET offrent des APIs pour les clients Amazon SQS. Ils vous permettent de travailler avec les fonctionnalités d'Amazon SQS telles que les files d'attente et les messages. Cette section contient un petit nombre d'exemples qui vous montrent les modèles que vous pouvez suivre lorsque vous travaillez avec ces outils APIs. Pour voir l'ensemble complet de APIs, consultez la [référence de l'AWS SDK pour .NET API](#) (et faites défiler la page jusqu'à « Amazon.sqs »).

Les Amazon SQS APIs sont fournis par le package [AWSSDK NuGet .SQS](#).

Conditions préalables

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

Rubriques

- [Création de files d'attente Amazon SQS](#)
- [Mise à jour des files d'attente Amazon SQS](#)
- [Suppression des files d'attente Amazon SQS](#)
- [Envoi de messages Amazon SQS](#)
- [Réception de messages Amazon SQS](#)

Création de files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le SDK pour AWS SDK pour .NET pour créer une file d'attente Amazon SQS. L'application crée une [file d'attente contenant des lettres mortes](#) si vous n'en

fournissez pas l'ARN. Il crée ensuite une file de messages standard, qui inclut une file de lettres mortes (celle que vous avez fournie ou celle qui a été créée).

Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement des informations sur toutes les files d'attente existantes.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Afficher les files d'attente existantes](#)
- [Création de la file d'attente](#)
- [Obtenir l'ARN d'une file d'attente](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Afficher les files d'attente existantes

L'extrait suivant présente une liste des files d'attente existantes dans la région du client SQS et les attributs de chaque file d'attente.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to show a list of the existing queues
private static async Task ShowQueues(IAmazonSQS sqsClient)
{
    ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
    Console.WriteLine();
    foreach(string qUrl in responseList.QueueUrls)
    {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
    }
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
```

```

var attributes = new List<string>{ QueueAttributeName.All };
GetQueueAttributesResponse responseGetAtt =
    await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
Console.WriteLine($"Queue: {qUrl}");
foreach(var att in responseGetAtt.Attributes)
    Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

```

Création de la file d'attente

L'extrait de code suivant crée une file d'attente. L'extrait inclut l'utilisation d'une file d'attente de lettres mortes, mais une file d'attente de lettres mortes n'est pas nécessairement requise pour vos files d'attente.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```

//
// Method to create a queue. Returns the queue URL.
private static async Task<string> CreateQueue(
    IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
    string maxReceiveCount=null, string receiveWaitTime=null)
{
    var attrs = new Dictionary<string, string>();

    // If a dead-letter queue is given, create a message queue
    if(!string.IsNullOrEmpty(deadLetterQueueUrl))
    {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
            $"{{\"deadLetterTargetArn\": \"{await GetQueueArn(sqsClient, " +
            deadLetterQueueUrl)}\"}}, \" +
            $"\"maxReceiveCount\": \"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
    }

    // If no dead-letter queue is given, create one of those instead
    //else
    //{
    // // Add attributes for the dead-letter queue as needed
    // attrs.Add();
    //}

    // Create the queue

```

```
CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(  
    new CreateQueueRequest{QueueName = qName, Attributes = attrs});  
return responseCreate.QueueUrl;  
}
```

Obtenir l'ARN d'une file d'attente

L'extrait suivant obtient l'ARN de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to get the ARN of a queue  
private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)  
{  
    GetQueueAttributesResponse responseGetAtt = await  
sqsClient.GetQueueAttributesAsync(  
    qUrl, new List<string>{QueueAttributeName.QueueArn});  
    return responseGetAtt.QueueARN;  
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.SQS](#)

Éléments de programmation :

- [Espace de noms Amazon.sqs](#)

Classe [Amazon SQSClient](#)

Classe [QueueAttributeName](#)

- [Espace de noms Amazon.SQS.Model](#)

Classe [CreateQueueRequest](#)

Classe [CreateQueueResponse](#)

Classe [GetQueueAttributesResponse](#)

Classe [ListQueuesResponse](#)

Le code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
    // = = = = =
    // Class to create a queue
    class Program
    {
        private const string MaxReceiveCount = "10";
        private const string ReceiveMessageWaitTime = "2";
        private const int MaxArgs = 3;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count > MaxArgs)
                CommandLine.ErrorExit(
                    "\nToo many command-line arguments.\nRun the command with no arguments to see help.");

            // Create the Amazon SQS client
            var sqsClient = new AmazonSQSClient();

            // In the case of no command-line arguments, just show help and the existing queues
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
            }
        }
    }
}
```

```

        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):");
    };

    string response = Console.ReadLine();
    if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
        await ShowQueues(sqsClient);
    return;
}

// Get the application arguments from the parsed list
string queueName =
    CommandLine.GetArgument(parsedArgs, null, "-q", "--queue-name");
string deadLetterQueueUrl =
    CommandLine.GetArgument(parsedArgs, null, "-d", "--dead-letter-queue");
string maxReceiveCount =
    CommandLine.GetArgument(parsedArgs, MaxReceiveCount, "-m", "--max-receive-
count");
string receiveWaitTime =
    CommandLine.GetArgument(parsedArgs, ReceiveMessageWaitTime, "-w", "--wait-
time");

if(string.IsNullOrEmpty(queueName))
    CommandLine.ErrorExit(
        "\nYou must supply a queue name.\nRun the command with no arguments to see
help.");

// If a dead-letter queue wasn't given, create one
if(string.IsNullOrEmpty(deadLetterQueueUrl))
{
    Console.WriteLine("\nNo dead-letter queue was specified. Creating one...");
    deadLetterQueueUrl = await CreateQueue(sqsClient, queueName + "__dlq");
    Console.WriteLine($"Your new dead-letter queue:");
    await ShowAllAttributes(sqsClient, deadLetterQueueUrl);
}

// Create the message queue
string messageQueueUrl = await CreateQueue(
    sqsClient, queueName, deadLetterQueueUrl, maxReceiveCount, receiveWaitTime);
Console.WriteLine($"Your new message queue:");
await ShowAllAttributes(sqsClient, messageQueueUrl);
}

//

```

```

// Method to show a list of the existing queues
private static async Task ShowQueues(IAmazonSQS sqsClient)
{
    ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
    Console.WriteLine();
    foreach(string qUrl in responseList.QueueUrls)
    {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
    }
}

//
// Method to create a queue. Returns the queue URL.
private static async Task<string> CreateQueue(
    IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
    string maxReceiveCount=null, string receiveWaitTime=null)
{
    var attrs = new Dictionary<string, string>();

    // If a dead-letter queue is given, create a message queue
    if(!string.IsNullOrEmpty(deadLetterQueueUrl))
    {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
            $"{{\"deadLetterTargetArn\": \"{await GetQueueArn(sqsClient, " +
deadLetterQueueUrl)}\"},\" +
            $"\"maxReceiveCount\": \"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
    }

    // If no dead-letter queue is given, create one of those instead
    //else
    //{
    // // Add attributes for the dead-letter queue as needed
    // attrs.Add();
    //}

    // Create the queue
    CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
        new CreateQueueRequest{QueueName = qName, Attributes = attrs});
    return responseCreate.QueueUrl;
}

```

```

//
// Method to get the ARN of a queue
private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
    qUrl, new List<string>{QueueAttributeName.QueueArn});
    return responseGetAtt.QueueARN;
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
    var attributes = new List<string>{ QueueAttributeName.All };
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
        "\\n -q, --queue-name: The name of the queue you want to create." +
        "\\n -d, --dead-letter-queue: The URL of an existing queue to be used as the
dead-letter queue."+
        "\\n      If this argument isn't supplied, a new dead-letter queue will be
created." +
        "\\n -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
of the queue." +
        $"\\n      Default is {MaxReceiveCount}." +
        "\\n -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
for long polling." +
        $"\\n      Default is {ReceiveMessageWaitTime}.");
}

```



```

    }
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];
                var value = key;

                // Check to see if there's a value that goes with this option?
                if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
                parsedArgs.Add(key, value);
            }

            // If the first argument in this iteration doesn't start with a dash, it's a
            value
            else

```

```

        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}
}

```

Considérations supplémentaires

- Le nom de votre file d'attente doit être composé de caractères alphanumériques, de traits d'union et de traits de soulignement.

- Les noms et les files d'attente URLs font la distinction majuscules/minuscules
- Si vous avez besoin de l'URL de la file d'attente mais que vous n'avez que le nom de la file d'attente, utilisez l'une `AmazonSQSClient.GetQueueUrlAsync` des méthodes.
- Pour plus d'informations sur les différents attributs de file d'attente que vous pouvez définir, consultez [CreateQueueRequest](#) la [référence d'AWS SDK pour .NET API](#) ou [SetQueueAttributes](#) la [référence d'API Amazon Simple Queue Service](#).
- Cet exemple indique une longue période d'interrogation pour tous les messages de la file d'attente que vous créez. Cela se fait à l'aide de l'`ReceiveMessageWaitTimeSeconds` attribut.

Vous pouvez également spécifier un long sondage lors d'un appel aux `ReceiveMessageAsync` méthodes de la `SQSClient` classe [Amazon](#). Pour de plus amples informations, veuillez consulter [Réception de messages Amazon SQS](#).

Pour plus d'informations sur les sondages courts par rapport aux sondages longs, consultez la section Sondage [court et long](#) du manuel Amazon Simple Queue Service Developer Guide.

- Une file d'attente de lettres mortes est une file que les autres files d'attente (source) peuvent cibler pour les messages qui ne sont pas traités correctement. Pour plus d'informations, consultez les [files d'attente contenant des lettres mortes Amazon SQS](#) dans le guide du développeur Amazon Simple Queue Service.
- Vous pouvez également consulter la liste des files d'attente et les résultats de cet exemple dans la console [Amazon SQS](#).

Mise à jour des files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le pour mettre AWS SDK pour .NET à jour une file d'attente Amazon SQS. Après quelques vérifications, l'application met à jour l'attribut donné avec la valeur donnée, puis affiche tous les attributs de la file d'attente.

Si seule l'URL de la file d'attente est incluse dans les arguments de la ligne de commande, l'application affiche simplement tous les attributs de la file d'attente.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Afficher les attributs de file d'attente](#)
- [Valider le nom d'attribut](#)
- [Mettre à jour l'attribut de file](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Afficher les attributs de file d'attente

L'extrait suivant montre les attributs de la file d'attente identifiés par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to show all attributes of a queue  
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)  
{  
    GetQueueAttributesResponse responseGetAtt =  
        await sqsClient.GetQueueAttributesAsync(qUrl,  
            new List<string>{ QueueAttributeName.All });  
    Console.WriteLine($"Queue: {qUrl}");  
    foreach(var att in responseGetAtt.Attributes)  
        Console.WriteLine($"{att.Key}: {att.Value}");  
}
```

Valider le nom d'attribut

L'extrait suivant valide le nom de l'attribut mis à jour.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to check the name of the attribute  
private static bool ValidAttribute(string attribute)  
{  
    var attOk = false;
```

```
var qAttNameType = typeof(QueueAttributeName);
List<string> qAttNamefields = new List<string>();
foreach(var field in qAttNameType.GetFields())
    qAttNamefields.Add(field.Name);
foreach(var name in qAttNamefields)
    if(attribute == name) { attOk = true; break; }
return attOk;
}
```

Mettre à jour l'attribut de file

L'extrait suivant met à jour un attribut de la file d'attente identifié par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to update a queue attribute
private static async Task UpdateAttribute(
    IAmazonSQS sqsClient, string qUrl, string attribute, string value)
{
    await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.SQS](#)

Éléments de programmation :

- [Espace de noms Amazon.sqs](#)

Classe [Amazon SQSClient](#)

Classe [QueueAttributeName](#)

- [Espace de noms Amazon.SQS.Model](#)

Classe [GetQueueAttributesResponse](#)

Le code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
    // = = = = =
    // Class to update a queue
    class Program
    {
        private const int MaxArgs = 3;
        private const int InvalidArgCount = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }
            if((parsedArgs.Count > MaxArgs) || (parsedArgs.Count == InvalidArgCount))
                CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Get the application arguments from the parsed list
            var qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
            var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
            var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

            if(string.IsNullOrEmpty(qUrl))
                CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
                    "\nRun the command with no arguments to see help.");
        }
    }
}
```

```

// Create the Amazon SQS client
var sqsClient = new AmazonSQSClient();

// In the case of one command-line argument, just show the attributes for the
queue
if(parsedArgs.Count == 1)
    await ShowAllAttributes(sqsClient, qUrl);

// Otherwise, attempt to update the given queue attribute with the given value
else
{
    // Check to see if the attribute is valid
    if(ValidAttribute(attribute))
    {
        // Perform the update and then show all the attributes of the queue
        await UpdateAttribute(sqsClient, qUrl, attribute, value);
        await ShowAllAttributes(sqsClient, qUrl);
    }
    else
    {
        Console.WriteLine($"\\nThe given attribute name, {attribute}, isn't valid.");
    }
}
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
            new List<string>{ QueueAttributeName.All });
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

//
// Method to check the name of the attribute
private static bool ValidAttribute(string attribute)
{

```

```

        var attOk = false;
        var qAttNameType = typeof(QueueAttributeName);
        List<string> qAttNamefields = new List<string>();
        foreach(var field in qAttNameType.GetFields())
            qAttNamefields.Add(field.Name);
        foreach(var name in qAttNamefields)
            if(attribute == name) { attOk = true; break; }
        return attOk;
    }

    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
        IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
        await sqsClient.SetQueueAttributesAsync(qUrl,
            new Dictionary<string, string>{{attribute, value}});
    }

    //
    // Command-line help
    private static void PrintHelp()
    {
        Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
        Console.WriteLine("  -q: The URL of the queue you want to update.");
        Console.WriteLine("  -a: The name of the attribute to update.");
        Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //

```



```

// Parameters:
// - args: The command-line arguments passed into the application by the system.
//
// Returns:
// A Dictionary with string Keys and Values.
//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:

```

```
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}
```

Considérations supplémentaires

- Pour mettre à jour l'`RedrivePolicy` attribut, vous devez mettre la valeur entière entre guillemets et éviter les guillemets pour les key/value paires, en fonction de votre système d'exploitation.

Sous Windows, par exemple, la valeur est construite de la manière suivante :

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\", \"maxReceiveCount\":\"10\"}"
```

Suppression des files d'attente Amazon SQS

Cet exemple vous montre comment utiliser le pour AWS SDK pour .NET supprimer une file d'attente Amazon SQS. L'application supprime la file d'attente, attend qu'elle disparaisse pendant un certain temps, puis affiche la liste des files d'attente restantes.

Si vous ne fournissez aucun argument de ligne de commande, l'application affiche simplement une liste des files d'attente existantes.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Supprimer la file d'attente](#)
- [Attendez que la file d'attente soit terminée](#)
- [Afficher la liste des files d'attente existantes](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Supprimer la file d'attente

L'extrait suivant supprime la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to delete an SQS queue  
private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)  
{  
    Console.WriteLine($"Deleting queue {qUrl}...");  
    await sqsClient.DeleteQueueAsync(qUrl);  
    Console.WriteLine($"Queue {qUrl} has been deleted.");  
}
```

Attendez que la file d'attente soit terminée

L'extrait suivant attend la fin du processus de suppression, ce qui peut prendre 60 secondes.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to wait up to a given number of seconds  
private static async Task Wait(  
    IAmazonSQS sqsClient, int numSeconds, string qUrl)  
{
```

```

    Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
    Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
    for(int i=0; i<numSeconds; i++)
    {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
            if(url == qUrl)
            {
                found = true;
                break;
            }
        }
        if(!found) break;
    }
}

```

Afficher la liste des files d'attente existantes

L'extrait suivant présente une liste des files d'attente existantes dans la région du client SQS.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```

//
// Method to show a list of the existing queues
private static async Task ListQueues(IAmazonSQS sqsClient)
{
    ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
    Console.WriteLine("\nList of queues:");
    foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
}

```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.SQS](#)

Éléments de programmation :

- [Espace de noms Amazon.sqs](#)

Classe [Amazon SQSClient](#)

- [Espace de noms Amazon.SQS.Model](#)

Classe [ListQueuesResponse](#)

Le code

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSDeleteQueue
{
    // = = = = =
    // Class to update a queue
    class Program
    {
        private const int TimeToWait = 60;

        static async Task Main(string[] args)
        {
            // Create the Amazon SQS client
            var sqsClient = new AmazonSQSClient();

            // If no command-line arguments, just show a list of the queues
            if(args.Length == 0)
            {
                Console.WriteLine("\nUsage: SQSCreateQueue queue_url");
                Console.WriteLine("    queue_url - The URL of the queue you want to delete.");
            }
        }
    }
}
```

```

        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):");
    };

    var response = Console.ReadLine();
    if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
        await ListQueues(sqsClient);
    return;
}

// If given a queue URL, delete that queue
if(args[0].StartsWith("https://sqs."))
{
    // Delete the queue
    await DeleteQueue(sqsClient, args[0]);
    // Wait for a little while because it takes a while for the queue to disappear
    await Wait(sqsClient, TimeToWait, args[0]);
    // Show a list of the remaining queues
    await ListQueues(sqsClient);
}
else
{
    Console.WriteLine("The command-line argument isn't a queue URL:");
    Console.WriteLine($"{args[0]}");
}
}

//
// Method to delete an SQS queue
private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($"Deleting queue {qUrl}...");
    await sqsClient.DeleteQueueAsync(qUrl);
    Console.WriteLine($"Queue {qUrl} has been deleted.");
}

//
// Method to wait up to a given number of seconds
private static async Task Wait(
    IAmazonSQS sqsClient, int numSeconds, string qUrl)
{
    Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
}

```

```

        Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
        for(int i=0; i<numSeconds; i++)
        {
            Console.Write(".");
            Thread.Sleep(1000);
            if(Console.KeyAvailable) break;

            // Check to see if the queue is gone yet
            var found = false;
            ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
            foreach(var url in responseList.QueueUrls)
            {
                if(url == qUrl)
                {
                    found = true;
                    break;
                }
            }
            if(!found) break;
        }
    }

    //
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        Console.WriteLine("\nList of queues:");
        foreach(var qUrl in responseList.QueueUrls)
            Console.WriteLine($"- {qUrl}");
    }
}

```

Considérations supplémentaires

- L'appel `DeleteQueueAsync` d'API ne vérifie pas si la file d'attente que vous supprimez est utilisée comme file d'attente de lettres mortes. Une procédure plus sophistiquée pourrait vérifier cela.

- Vous pouvez également consulter la liste des files d'attente et les résultats de cet exemple dans la console [Amazon SQS](#).

Envoi de messages Amazon SQS

[Cet exemple vous montre comment utiliser le pour envoyer des messages AWS SDK pour .NET à une file d'attente Amazon SQS, que vous pouvez créer par programmation ou à l'aide de la console Amazon SQS](#). L'application envoie un seul message à la file d'attente, puis un lot de messages. L'application attend ensuite l'entrée de l'utilisateur, qui peut prendre la forme de messages supplémentaires à envoyer à la file d'attente ou d'une demande de sortie de l'application.

Cet exemple et le [suivant concernant la réception de messages](#) peuvent être utilisés conjointement pour voir le flux des messages dans Amazon SQS.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Envoyer un message](#)
- [Envoyer un lot de messages](#)
- [Supprimer tous les messages de la file](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Envoyer un message

L'extrait suivant envoie un message à la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to put a message on a queue  
// Could be expanded to include message attributes, etc., in a SendMessageRequest  
private static async Task SendMessage(  
    IAmazonSQS sqsClient, string qUrl, string messageBody)  
{  
    SendMessageResponse responseSendMsg =  
        await sqsClient.SendMessageAsync(qUrl, messageBody);  
}
```



```
Console.WriteLine($"Message added to queue\n {qUrl}");
Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}
```

Envoyer un lot de messages

L'extrait suivant envoie un lot de messages à la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
    Console.WriteLine($" \nSending a batch of messages to queue\n {qUrl}");
    SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
}
```

Supprimer tous les messages de la file

L'extrait suivant supprime tous les messages de la file d'attente identifiée par l'URL de file d'attente donnée. Cette opération est également connue sous le nom de purge de la file d'attente.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($" \nPurging messages from queue\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.SQS](#)

Éléments de programmation :

- [Espace de noms Amazon.sqs](#)

Classe [Amazon SQSClient](#)

- [Espace de noms Amazon.SQS.Model](#)

Classe [PurgeQueueResponse](#)

Classe [SendMessageBatchResponse](#)

Classe [SendMessageResponse](#)

Classe [SendMessageBatchRequestEntry](#)

Classe [SendMessageBatchResultEntry](#)

Le code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
    // = = = = =
    // Class to send messages to a queue
    class Program
    {
```

```

// Some example messages to send to the queue
private const string JsonMessage = "{\"product\":{\"name\":\"Product A\",\"price\
\": \"32\"},{\"name\": \"Product B\",\"price\": \"27\"}}";
private const string XmlMessage = "<products><product name=\"Product A\" price=
\"32\" /><product name=\"Product B\" price=\"27\" /></products>";
private const string CustomMessage = "||product|Product A|32||product|Product B|
27||";
private const string TextMessage = "Just a plain text message.";

static async Task Main(string[] args)
{
    // Do some checks on the command-line
    if(args.Length == 0)
    {
        Console.WriteLine("\nUsage: SQSSendMessages queue_url");
        Console.WriteLine("    queue_url - The URL of an existing SQS queue.");
        return;
    }
    if(!args[0].StartsWith("https://sqs."))
    {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
    }

    // Create the Amazon SQS client
    var sqsClient = new AmazonSQSClient();

    // (could verify that the queue exists)
    // Send some example messages to the given queue
    // A single message
    await SendMessage(sqsClient, args[0], JsonMessage);

    // A batch of messages
    var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
    await SendMessageBatch(sqsClient, args[0], batchMessages);

    // Let the user send their own messages or quit
    await InteractWithUser(sqsClient, args[0]);

    // Delete all messages that are still in the queue

```

```

    await DeleteAllMessages(sqsClient, args[0]);
}

//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
{
    SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}

//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
    Console.WriteLine($"Sending a batch of messages to queue\n {qUrl}");
    SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
}

//
// Method to get input from the user
// They can provide messages to put in the queue or exit the application
private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
{
    string response;
    while (true)
    {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
    }
}

```

```
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
    }
}

//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($"\\nPurging messages from queue\\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
}
```

Considérations supplémentaires

- Pour plus d'informations sur les différentes limitations relatives aux messages, y compris les caractères autorisés, consultez la section [Quotas relatifs aux messages](#) dans le manuel [Amazon Simple Queue Service Developer Guide](#).
- Les messages restent dans les files d'attente jusqu'à ce qu'ils soient supprimés ou que la file d'attente soit purgée. Lorsqu'un message est reçu par une application, il n'est pas visible dans la file d'attente même s'il existe toujours dans la file d'attente. Pour plus d'informations sur les délais de visibilité, consultez [Amazon SQS Visibility Timeout](#).
- Outre le corps du message, vous pouvez également ajouter des attributs aux messages. Pour plus d'informations, consultez la section [Métadonnées des messages](#).

Réception de messages Amazon SQS

[Cet exemple vous montre comment utiliser le AWS SDK pour .NET pour recevoir des messages d'une file d'attente Amazon SQS, que vous pouvez créer par programmation ou à l'aide de la console Amazon SQS](#). L'application lit un seul message dans la file d'attente, traite le message (dans ce cas,

affiche le corps du message sur la console), puis le supprime de la file d'attente. L'application répète ces étapes jusqu'à ce que l'utilisateur tape une touche sur le clavier.

Cet exemple et l'[exemple précédent concernant l'envoi de messages](#) peuvent être utilisés conjointement pour voir le flux des messages dans Amazon SQS.

Les sections suivantes fournissent des extraits de cet exemple. Le [code complet de l'exemple](#) est affiché ensuite et peut être créé et exécuté tel quel.

Rubriques

- [Recevez un message](#)
- [Supprimer un message](#)
- [Code complet](#)
- [Considérations supplémentaires](#)

Recevez un message

L'extrait suivant reçoit un message de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to read a message from the given queue  
// In this example, it gets one message at a time  
private static async Task<ReceiveMessageResponse> GetMessage(  
    IAmazonSQS sqsClient, string qUrl, int waitTime=0)  
{  
    return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{  
        QueueUrl=qUrl,  
        MaxNumberOfMessages=MaxMessages,  
        WaitTimeSeconds=waitTime  
        // (Could also request attributes, set visibility timeout, etc.)  
    });  
}
```

Supprimer un message

L'extrait suivant supprime un message de la file d'attente identifiée par l'URL de file d'attente donnée.

L'exemple [à la fin de cette rubrique montre cet](#) extrait de code en cours d'utilisation.

```
//  
// Method to delete a message from a queue  
private static async Task DeleteMessage(  
    IAmazonSQS sqsClient, Message message, string qUrl)  
{  
    Console.WriteLine($"\\nDeleting message {message.MessageId} from queue...");  
    await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);  
}
```

Code complet

Cette section présente les références pertinentes et le code complet de cet exemple.

Références du SDK

NuGet colis :

- [AWSSDK.SQS](#)

Éléments de programmation :

- [Espace de noms Amazon.sqs](#)

Classe [Amazon SQSClient](#)

- [Espace de noms Amazon.SQS.Model](#)

Classe [ReceiveMessageRequest](#)

Classe [ReceiveMessageResponse](#)

Le code

```
using System;  
using System.Threading.Tasks;  
using Amazon.SQS;  
using Amazon.SQS.Model;  
  
namespace SQSReceiveMessages  
{  
    class Program  
    {
```

```

private const int MaxMessages = 1;
private const int WaitTime = 2;
static async Task Main(string[] args)
{
    // Do some checks on the command-line
    if(args.Length == 0)
    {
        Console.WriteLine("\nUsage: SQSReceiveMessages queue_url");
        Console.WriteLine("    queue_url - The URL of an existing SQS queue.");
        return;
    }
    if(!args[0].StartsWith("https://sqs."))
    {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
    }

    // Create the Amazon SQS client
    var sqsClient = new AmazonSQSClient();

    // (could verify that the queue exists)
    // Read messages from the queue and perform appropriate actions
    Console.WriteLine($"Reading messages from queue\n {args[0]}");
    Console.WriteLine("Press any key to stop. (Response might be slightly
delayed.)");
    do
    {
        var msg = await GetMessage(sqsClient, args[0], WaitTime);
        if(msg.Messages.Count != 0)
        {
            if(ProcessMessage(msg.Messages[0]))
                await DeleteMessage(sqsClient, msg.Messages[0], args[0]);
        }
    } while(!Console.KeyAvailable);
}

//
// Method to read a message from the given queue
// In this example, it gets one message at a time
private static async Task<ReceiveMessageResponse> GetMessage(
    IAmazonSQS sqsClient, string qUrl, int waitTime=0)
{

```



```

    return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
        QueueUrl=qUrl,
        MaxNumberOfMessages=MaxMessages,
        WaitTimeSeconds=waitTime
        // (Could also request attributes, set visibility timeout, etc.)
    });
}

//
// Method to process a message
// In this example, it simply prints the message
private static bool ProcessMessage(Message message)
{
    Console.WriteLine($"\\nMessage body of {message.MessageId}:");
    Console.WriteLine($"{message.Body}");
    return true;
}

//
// Method to delete a message from a queue
private static async Task DeleteMessage(
    IAmazonSQS sqsClient, Message message, string qUrl)
{
    Console.WriteLine($"\\nDeleting message {message.MessageId} from queue...");
    await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
}
}
}

```

Considérations supplémentaires

- Pour spécifier un long sondage, cet exemple utilise la `WaitTimeSeconds` propriété pour chaque appel à la `ReceiveMessageAsync` méthode.

Vous pouvez également spécifier un long sondage pour tous les messages d'une file d'attente en utilisant l'`ReceiveMessageWaitTimeSeconds` attribut lors de la [création](#) ou de la [mise à jour](#) de la file d'attente.

Pour plus d'informations sur les sondages courts par rapport aux sondages longs, consultez la section Sondage [court et long](#) du manuel Amazon Simple Queue Service Developer Guide.

- Pendant le traitement des messages, vous pouvez utiliser le descripteur de réception pour modifier le délai de visibilité des messages. Pour plus d'informations sur la procédure à suivre, consultez les [ChangeMessageVisibilityAsync](#) méthodes de la [SQSClient](#) classe [Amazon](#).
- L'appel de la [DeleteMessageAsync](#) méthode supprime le message de la file d'attente de manière inconditionnelle, quel que soit le paramètre de délai d'expiration de visibilité.

Utilisation AWS Lambda pour le service de calcul

Les AWS SDK pour .NET supports AWS Lambda, qui vous permettent d'exécuter du code sans provisionner ni gérer de serveurs. Pour plus d'informations, consultez la [page AWS Lambda du produit](#) et le [guide du AWS Lambda développeur](#), en particulier la section consacrée à l'[utilisation de C#](#).

APIs

Le AWS SDK pour .NET APIs prévoit AWS Lambda. [Ils vous APIs permettent d'utiliser les fonctionnalités Lambda telles que les fonctions, les déclencheurs et les événements](#). Pour voir l'ensemble complet de APIs, consultez [Lambda](#) dans la référence de l'[AWS SDK pour .NET API](#).

[Les Lambda APIs sont fournis par NuGet des packages.](#)

Prérequis

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Informations supplémentaires

Consultez [Intégration AWS à .NET Aspire](#) pour plus d'informations sur le développement AWS Lambda avec .NET Aspire.

Rubriques

Rubriques

- [Utiliser des annotations pour écrire des fonctions AWS Lambda](#)

Utiliser des annotations pour écrire des fonctions AWS Lambda

Lorsque vous écrivez des fonctions Lambda, vous devez parfois écrire une grande quantité de code de gestionnaire et mettre à jour des AWS CloudFormation modèles, entre autres tâches. Lambda Annotations est un framework destiné à alléger ces charges pour les fonctions Lambda de .NET 6, rendant ainsi l'expérience d'écriture Lambda plus naturelle en C#.

À titre d'exemple des avantages de l'utilisation du framework Lambda Annotations, considérez les extraits de code suivants qui ajoutent deux nombres.

Sans annotations Lambda

```
public class Functions
{
    public APIGatewayProxyResponse LambdaMathPlus(APIGatewayProxyRequest request,
        ILambdaContext context)
    {
        if (!request.PathParameters.TryGetValue("x", out var xs))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }
        if (!request.PathParameters.TryGetValue("y", out var ys))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }

        var x = int.Parse(xs);
        var y = int.Parse(ys);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = (x + y).ToString(),
            Headers = new Dictionary<string, string> { { "Content-Type", "text/
plain" } }
        };
    }
}
```

```
}
```

Avec des annotations Lambda

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Comme le montre l'exemple, les annotations Lambda peuvent supprimer le besoin d'un certain code de plaque de cuisson.

Pour plus de détails sur l'utilisation du framework ainsi que des informations supplémentaires, consultez les ressources suivantes :

- Le [GitHub fichier README](#) pour la documentation sur les attributs APIs et les attributs des annotations Lambda.
- L'[article de blog](#) sur les annotations Lambda.
- Le package [Amazon.Lambda.Annotations](#) NuGet .
- Le [projet Photo Asset Management](#) sur GitHub. [Plus précisément, consultez le PamApiAnnotationsdossier et les références aux annotations Lambda dans le fichier README du projet.](#)

Bibliothèques et frameworks de haut niveau pour AWS SDK pour .NET

Les sections suivantes contiennent des informations sur les bibliothèques et les frameworks de haut niveau qui ne font pas partie des fonctionnalités principales du SDK. Ces bibliothèques et frameworks utilisent les fonctionnalités de base du SDK pour créer des fonctionnalités qui facilitent certaines tâches.

Si vous êtes nouveau dans le AWS SDK pour .NET domaine, vous voudrez peut-être d'abord consulter le [Faites une visite rapide](#) sujet. Il vous présente le SDK.

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Rubriques

- [AWS Framework de traitement des messages pour .NET](#)
- [Intégration AWS à .NET Aspire dans le AWS SDK pour .NET](#)

AWS Framework de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET est un framework AWS natif qui simplifie le développement d'applications de traitement de messages .NET utilisant AWS des services tels qu'Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) et Amazon EventBridge. Le framework réduit la quantité de code standard que les développeurs doivent écrire, ce qui vous permet de vous concentrer sur votre logique métier lors de la publication et de la consommation de messages. Pour plus de détails sur la façon dont le framework peut simplifier votre développement, consultez le billet de blog [Présentation du framework de traitement des AWS messages pour .NET \(version préliminaire\)](#). La première partie en particulier fournit une démonstration qui montre la différence entre l'utilisation d'appels d'API de bas niveau et l'utilisation du framework.

Le cadre de traitement des messages prend en charge les activités et fonctionnalités suivantes :

- Envoi de messages à SQS et publication d'événements sur SNS et EventBridge
- Réception et gestion des messages de SQS à l'aide d'un sondeur de longue durée, généralement utilisé dans les services d'arrière-plan. Cela inclut la gestion du délai de visibilité pendant le traitement d'un message afin d'empêcher d'autres clients de le traiter.
- Gestion des messages dans AWS Lambda les fonctions.
- FIFO (first-in-first-out), files d'attente SQS et sujets SNS.
- OpenTelemetry pour la journalisation.

Pour plus de détails sur ces activités et fonctionnalités, consultez la section Fonctionnalités du [billet de blog](#) et les sujets répertoriés ci-dessous.

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Ressources supplémentaires

- Le [AWS.Messaging](#) package sur [NuGet.org](#).
- La [référence de l'API](#).
- Le README fichier dans le GitHub dépôt à l'adresse <https://github.com/aws/aws-dotnet-messaging/>
- [Injection de dépendances .NET](#) par Microsoft.
- [.NET Generic Host](#) de Microsoft.

Rubriques

- [Commencez avec le cadre de traitement des AWS messages pour .NET](#)
- [Publiez des messages avec le AWS cadre de traitement des messages pour .NET](#)
- [Consommez des messages avec le AWS cadre de traitement des messages pour .NET](#)
- [Utilisation du FIFO avec le cadre de traitement des AWS messages pour .NET](#)
- [Journalisation et télémétrie ouverte pour le cadre de traitement des AWS messages pour .NET](#)
- [Personnalisation de l'infrastructure de traitement des AWS messages pour .NET](#)
- [Sécurité de l'infrastructure de traitement des AWS messages pour .NET](#)

Commencez avec le cadre de traitement des AWS messages pour .NET

Avant de commencer, assurez-vous d'avoir [configuré votre environnement et votre projet](#). Consultez également les informations contenues dans [Fonctionnalités du SDK](#).

Cette rubrique fournit des informations qui vous aideront à commencer à utiliser le cadre de traitement des messages. Outre les informations relatives aux prérequis et à la configuration, un didacticiel est fourni pour vous montrer comment implémenter un scénario courant.

Prérequis et configuration

- Les informations d'identification que vous fournissez pour votre application doivent disposer des autorisations appropriées pour le service de messagerie et les opérations qu'il utilise. Pour plus d'informations, consultez les rubriques relatives à la sécurité pour [SQS](#) et [SNS](#), ainsi que [EventBridge](#) dans leurs guides de développement respectifs. Consultez également la partie du fichier [README](#) GitHub qui traite des [autorisations](#) spécifiques.
- Pour utiliser le cadre de traitement des AWS messages pour .NET, vous devez ajouter le [AWS.Messaging](#) NuGetpackage à votre projet. Par exemple :

```
dotnet add package AWS.Messaging
```

- Le framework s'intègre au [conteneur de services d'injection de dépendances \(DI\) de .NET](#). Vous pouvez configurer le framework lors du démarrage de votre application en appelant `AddAWSMessageBus` pour l'ajouter au conteneur DI.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

didacticiel

Ce didacticiel explique comment utiliser le cadre de traitement des AWS messages pour .NET. Elle crée deux applications : une API ASP.NET Core Minimal qui envoie des messages à une file d'attente Amazon SQS lorsqu'elle reçoit une demande sur un point de terminaison d'API, et une application de console de longue durée qui interroge ces messages et les gère.

- Les instructions de ce didacticiel privilégient l'interface de ligne de commande .NET, mais vous pouvez exécuter ce didacticiel à l'aide d'outils multiplateformes tels que .NET CLI ou Microsoft Visual Studio. Pour plus d'informations sur les outils, voir [Installez et configurez votre chaîne d'outils](#).
- Ce didacticiel part du principe que vous utilisez votre [default] profil comme identifiant. Cela suppose également que des informations d'identification à court terme sont disponibles avec les autorisations appropriées pour envoyer et recevoir des messages Amazon SQS. Pour plus d'informations, consultez [Configurer l'authentification du SDK avec AWS](#) les rubriques relatives à la sécurité relatives à [SQS](#).

Note

L'exécution de ce didacticiel peut entraîner des coûts liés à la messagerie SQS.

Étapes

- [Création d'une file d'attente SQS](#)
- [Création et exécution de l'application de publication](#)
- [Création et exécution de l'application de gestion](#)
- [Nettoyage](#)

Création d'une file d'attente SQS

Ce didacticiel nécessite une file d'attente SQS pour envoyer et recevoir des messages. Une file d'attente peut être créée à l'aide de l'une des commandes suivantes pour le AWS CLI ou le Outils AWS pour PowerShell. Prenez note de l'URL de la file d'attente renvoyée afin de pouvoir la spécifier dans la configuration du framework qui suit.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Outils AWS pour PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Création et exécution de l'application de publication

Suivez la procédure ci-dessous pour créer et exécuter l'application de publication.

1. Ouvrez une invite de commande ou un terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

```
dotnet new webapi --name Publisher
```

3. Accédez au dossier du nouveau projet. Ajoutez une dépendance au cadre de traitement des AWS messages pour .NET.

```
cd Publisher  
dotnet add package AWS.Messaging
```


 Note

Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veuillez également à ajouter `AWSSDK.SSO` et `AWSSDK.SSOOIDC`.

4. Remplacez le code `Program.cs` par le code suivant.

```
using AWS.Messaging;
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/
// swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
    // The SQS URL should be passed as a command line argument or set in the Debug
    // launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

```

    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
    Publisher.GreetingMessage message) =>
    {
        return await PostGreeting(message, publisher);
    })
    .WithName("SendGreeting")
    .WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}

namespace Publisher
{
    /// <summary>

```

```
/// This class represents the message contents.  
/// </summary>  
public class GreetingMessage  
{  
    public string? SenderName { get; set; }  
    public string? Greeting { get; set; }  
}  
}
```

5. Exécutez la commande suivante. Cela devrait ouvrir une fenêtre de navigateur avec l'interface utilisateur Swagger, qui vous permettra d'explorer et de tester votre API.

```
dotnet watch run <queue URL created earlier>
```

6. Ouvrez le /greeting point de terminaison et choisissez Try it out.
7. Spécifiez greeting les valeurs senderName et les valeurs du message, puis choisissez Execute. Cela appelle votre API, qui envoie le message SQS.

Création et exécution de l'application de gestion

Utilisez la procédure suivante pour créer et exécuter l'application de gestion.

1. Ouvrez une invite de commande ou un terminal. Recherchez ou créez un dossier de système d'exploitation sous lequel vous pouvez créer un projet .NET.
2. Dans ce dossier, exécutez la commande suivante pour créer le projet .NET.

```
dotnet new console --name Handler
```

3. Accédez au dossier du nouveau projet. Ajoutez une dépendance au cadre de traitement des AWS messages pour .NET. Ajoutez également le Microsoft.Extensions.Hosting package, qui vous permet de configurer le framework via le [.NET Generic Host](#).

```
cd Handler  
dotnet add package AWS.Messaging  
dotnet add package Microsoft.Extensions.Hosting
```

 Note

Si vous l'utilisez AWS IAM Identity Center pour l'authentification, veuillez également à ajouter `AWSSDK.SSO` et `AWSSDK.SSO0IDC`.

4. Remplacez le code `Program.cs` par le code suivant.

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
        // Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);

            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
            GreetingMessage>("greetingMessage");
        }
        // You can add additional message handlers here, using different message
        // types.
    });
});

var host = builder.Build();
await host.RunAsync();
```

```
namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from {messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. Exécutez la commande suivante. Cela lance un sondage de longue haleine.

```
dotnet run <queue URL created earlier>
```

Peu après le démarrage, l'application recevra le message envoyé dans la première partie de ce didacticiel et enregistrera le message suivant :

```
Received message {greeting} from {senderName}
```

6. Appuyez Ctrl+C pour arrêter le sondeur.

Nettoyage

Utilisez l'une des commandes suivantes pour AWS CLI ou pour Outils AWS pour PowerShell supprimer la file d'attente.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Outils AWS pour PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publiez des messages avec le AWS cadre de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET permet de publier un ou plusieurs types de messages, de traiter un ou plusieurs types de messages ou d'effectuer les deux dans la même application.

Le code suivant montre la configuration d'une application qui publie différents types de messages vers différents AWS services.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
    EventBridge bus
    builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-
west-2:012345678910:event-bus/default");
});
```

Une fois que vous avez enregistré le framework au démarrage, injectez le générique `IMessagePublisher` dans votre code. Appelez sa `PublishAsync` méthode pour publier l'un des types de messages configurés ci-dessus. L'éditeur générique déterminera la destination vers laquelle acheminer le message en fonction de son type.

Dans l'exemple suivant, un contrôleur ASP.NET MVC reçoit à la fois `ChatMessage` des messages et des `OrderInfo` événements de la part des utilisateurs, puis les publie sur Amazon SQS et Amazon SNS respectivement. Les deux types de messages peuvent être publiés à l'aide de l'éditeur générique configuré ci-dessus.

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;

    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
```

```
{
    if (message == null)
    {
        return BadRequest("An order was not submitted.");
    }

    // Publish the OrderInfo to SNS, using the generic publisher.
    await _messagePublisher.PublishAsync(message);

    return Ok();
}
```

Afin d'acheminer un message vers la logique de traitement appropriée, le framework utilise des métadonnées appelées identifiant de type de message. Par défaut, il s'agit du nom complet du type .NET du message, y compris son nom d'assembly. Si vous envoyez et gérez à la fois des messages, ce mécanisme fonctionne bien si vous partagez la définition des objets de vos messages entre différents projets. Toutefois, si les messages sont redéfinis dans différents espaces de noms, ou si vous échangez des messages avec d'autres frameworks ou langages de programmation, vous devrez peut-être remplacer l'identifiant du type de message.

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
        // Register that you'll publish messages of type GreetingMessage to an existing
        queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

Éditeurs spécifiques aux services

L'exemple ci-dessus utilise le générique `IMessagePublisher`, qui peut être publié sur n'importe quel AWS service pris en charge en fonction du type de message configuré. Le framework fournit également des éditeurs spécifiques aux services pour Amazon SQS, Amazon SNS et Amazon EventBridge. Ces éditeurs spécifiques présentent des options qui ne s'appliquent

qu'à ce service et peuvent être injectées à l'aide des types `ISQSPublisher`, `ISNSPublisher`, et `IEventBridgePublisher`.

Par exemple, lorsque vous envoyez des messages à une file d'attente FIFO SQS, vous devez définir l'ID de [groupe de messages](#) approprié. Le code suivant montre à nouveau l'`ChatMessage` exemple, mais utilise maintenant un `ISQSPublisher` pour définir des options spécifiques à SQS.

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
            MessageDeduplicationId = <message-deduplication-id>,
            MessageGroupId = <message-group-id>
        });

        return Ok();
    }
}
```

La même chose peut être faite pour les réseaux sociaux et EventBridge, `IEventBridgePublisher` respectivement, en utilisant `ISNSPublisher` et.

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

Par défaut, les messages d'un type donné sont envoyés à la destination configurée à l'avance. Cependant, vous pouvez modifier la destination d'un seul message en utilisant les éditeurs spécifiques au message. Vous pouvez également remplacer le AWS SDK pour .NET client sous-jacent utilisé pour publier le message, ce qui peut être utile dans les applications mutualisées où vous devez modifier les rôles ou les informations d'identification, en fonction de la destination.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consommez des messages avec le AWS cadre de traitement des messages pour .NET

Le cadre de traitement des AWS messages pour .NET vous permet de consommer des messages [publiés](#) à l'aide du framework ou de l'un des services de messagerie. Les messages peuvent être consommés de différentes manières, dont certaines sont décrites ci-dessous.

Gestionnaires de messages

Pour consommer des messages, implémentez un gestionnaire de messages utilisant l'`IMessageHandler` interface pour chaque type de message que vous souhaitez traiter. Le mappage entre les types de messages et les gestionnaires de messages est configuré au démarrage du projet.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
            your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd");

            // Register all IMessageHandler implementations with the message type they
            should process.
            // Here messages that match our ChatMessage .NET type will be handled by
            our ChatMessageHandler
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Le code suivant montre un exemple de gestionnaire de messages pour un `ChatMessage` message.

```
public class ChatMessageHandler : IMessageHandler<ChatMessage>
{
    public Task<MessageProcessStatus> HandleAsync(MessageEnvelope<ChatMessage>
messageEnvelope, CancellationToken token = default)
    {
        // Add business and validation logic here.
        if (messageEnvelope == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        if (messageEnvelope.Message == null)
        {

```

```

        return Task.FromResult(MessageProcessStatus.Failed());
    }

    ChatMessage message = messageEnvelope.Message;

    Console.WriteLine($"Message Description: {message.MessageDescription}");

    // Return success so the framework will delete the message from the queue.
    return Task.FromResult(MessageProcessStatus.Success());
}
}

```

L'extérieur `MessageEnvelope` contient les métadonnées utilisées par le framework. Sa `message` propriété est le type de message (dans ce cas `ChatMessage`).

Vous pouvez revenir `MessageProcessStatus.Success()` pour indiquer que le message a été traité avec succès et que le framework supprimera le message de la file d'attente Amazon SQS. Au retour `MessageProcessStatus.Failed()`, le message restera dans la file d'attente où il pourra être traité à nouveau ou déplacé vers une [file d'attente contenant des lettres mortes](#), si cela est configuré.

Gestion des messages dans le cadre d'un processus de longue haleine

Vous pouvez appeler `AddSQSPoller` avec une URL de file d'attente SQS pour démarrer une longue série [BackgroundService](#) qui interrogera continuellement la file d'attente et traitera les messages.

```

await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
            your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
            {
                // The maximum number of messages from this queue that the framework
                will process concurrently on this client.
                options.MaxNumberOfConcurrentMessages = 10;
            }
        }
    }
    );

```

```
        // The duration each call to SQS will wait for new messages.
        options.WaitTimeSeconds = 20;
    });

    // Register all IMessageHandler implementations with the message type they
    should process.
    builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
    });
}
.Build()
.RunAsync();
```

Configuration du SQS Message Poller

Le sondeur de messages SQS peut être configuré par le `SQSMessagesPollerOptions` lors de l'appel `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- Le nombre maximum de messages de la file d'attente à traiter simultanément. La valeur par défaut est 10.
- `WaitTimeSeconds`- Durée (en secondes) pendant laquelle l'appel `ReceiveMessage` SQS attend l'arrivée d'un message dans la file d'attente avant de revenir. Si un message est disponible, l'appel revient plus tôt que prévu `WaitTimeSeconds`. La valeur par défaut est 20.

Gestion des délais de visibilité des messages

Les messages SQS ont un [délai d'expiration de visibilité](#). Lorsqu'un client commence à traiter un message donné, celui-ci reste dans la file d'attente, mais il est caché aux autres consommateurs pour éviter de le traiter plusieurs fois. Si le message n'est pas traité et supprimé avant de redevenir visible, un autre consommateur peut tenter de traiter le même message.

Le framework suivra et tentera de prolonger le délai de visibilité pour les messages qu'il gère actuellement. Vous pouvez configurer ce comportement `SQSMessagesPollerOptions` lors de l'appel `AddSQSPoller`.

- `VisibilityTimeout`- Durée en secondes pendant laquelle les messages reçus sont masqués lors des demandes de récupération ultérieures. La valeur par défaut est 30.
- `VisibilityTimeoutExtensionThreshold`- Lorsque le délai de visibilité d'un message est inférieur à ce nombre de secondes après son expiration, le cadre prolonge le délai de visibilité (de quelques secondes supplémentaires `VisibilityTimeout`). La valeur par défaut est 5.

- `VisibilityTimeoutExtensionHeartbeatInterval`- Fréquence en secondes à laquelle le framework vérifiera les messages arrivés à expiration dans les `VisibilityTimeoutExtensionThreshold` secondes qui suivent leur expiration, puis prolongera leur délai de visibilité. La valeur par défaut est 1.

Dans l'exemple suivant, le framework vérifiera toutes les secondes les messages toujours en cours de traitement. Pour les messages redevenus visibles 5 secondes, le framework prolongera automatiquement le délai de visibilité de chaque message de 30 secondes supplémentaires.

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
    options =>
    {
        options.VisibilityTimeout = 30;
        options.VisibilityTimeoutExtensionThreshold = 5;
        options.VisibilityTimeoutExtensionHeartbeatInterval = 1;
    });
```

Gestion des messages dans les AWS Lambda fonctions

Vous pouvez utiliser le cadre de traitement des AWS messages pour .NET avec [l'intégration de SQS à Lambda](#). Ceci est fourni par le `AWS.Messaging.Lambda` package. Reportez-vous à son [fichier README](#) pour commencer.

Utilisation du FIFO avec le cadre de traitement des AWS messages pour .NET

Pour les cas d'utilisation où l'ordre et la déduplication des messages sont essentiels, le cadre de traitement des AWS messages pour .NET prend en charge les files d'attente [Amazon SQS first-in-first-out](#) (FIFO) et les rubriques [Amazon SNS](#).

Publication

Lorsque vous publiez des messages dans une file d'attente ou une rubrique FIFO, vous devez définir l'ID du groupe de messages, qui indique le groupe auquel appartient le message. Les messages au sein d'un groupe sont traités dans l'ordre. Vous pouvez le définir sur les éditeurs de messages spécifiques à SQS et spécifiques au SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
```

```
MessageDeduplicationId = <message-deduplication-id>,  
MessageGroupId = <message-group-id>  
});
```

Abonnement en cours

Lors de la gestion des messages d'une file d'attente FIFO, le framework gère les messages d'un groupe de messages donné dans l'ordre dans lequel ils ont été reçus pour chaque `ReceiveMessages` appel. Le framework entre automatiquement dans ce mode de fonctionnement lorsqu'il est configuré avec une file d'attente se terminant par `.fifo`.

```
await Host.CreateDefaultBuilder(args)  
    .ConfigureServices(services =>  
    {  
        // Register the AWS Message Processing Framework for .NET.  
        services.AddAWSMessageBus(builder =>  
        {  
            // Because this is a FIFO queue, the framework automatically handles these  
            messages in order.  
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/  
MPF.fifo");  
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();  
        });  
    })  
    .Build()  
    .RunAsync();
```

Journalisation et télémétrie ouverte pour le cadre de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET est conçu OpenTelemetry pour enregistrer les [traces](#) de chaque message publié ou traité par le framework. Ceci est fourni par le [AWS.Messaging.Telemetry.OpenTelemetry](#) package. Reportez-vous à son [fichier README](#) pour commencer.

Note

Pour obtenir des informations de sécurité relatives à la journalisation, consultez [Sécurité de l'infrastructure de traitement des AWS messages pour .NET](#).

Personnalisation de l'infrastructure de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET crée, envoie et gère les messages dans trois « couches » différentes :

1. Au niveau de la couche la plus externe, le framework crée la requête ou la réponse AWS-native spécifique à un service. Avec Amazon SQS, par exemple, il crée des [SendMessage](#) requêtes et travaille avec les [Message](#) objets définis par le service.
2. [Dans la demande et la réponse SQS, le framework définit l'MessageBody élément \(ou Message pour Amazon SNS ou Detail pour EventBridge Amazon\) au format JSON.](#) [CloudEvent](#) Il contient les métadonnées définies par le framework qui sont accessibles sur l'MessageEnvelope objet lors de la gestion d'un message.
3. Au niveau de la couche la plus interne, l'`data` attribut à l'intérieur de l'objet CloudEvent JSON contient une sérialisation JSON de l'objet .NET envoyé ou reçu sous forme de message.

```
{
  "id": "b02f156b-0f02-48cf-ae54-4fbbe05cffba",
  "source": "/aws/messaging",
  "specversion": "1.0",
  "type": "Publisher.Models.ChatMessage",
  "time": "2023-11-21T16:36:02.8957126+00:00",
  "data": "<the ChatMessage object serialized as JSON>"
}
```

Vous pouvez personnaliser la façon dont l'enveloppe du message est configurée et lue :

- "id" identifie le message de manière unique. Par défaut, il est défini sur un nouveau GUID, mais celui-ci peut être remplacé en implémentant le vôtre `IMessageIdGenerator` et en l'injectant dans le conteneur DI.
- "type" contrôle la manière dont le message est acheminé vers les gestionnaires. Par défaut, le nom complet du type .NET correspondant au message est utilisé. Vous pouvez le remplacer via le `messageTypeIdentifieur` paramètre lorsque vous mappez le type de message à la destination via `AddSQSPublisher`, `AddSNSPublisher`, ou `AddEventBridgePublisher`.
- "source" indique le système ou le serveur qui a envoyé le message.
 - Il s'agira du nom de la fonction en cas de publication depuis AWS Lambda, du nom du cluster et de l'ARN de la tâche si vous utilisez Amazon ECS, de l'ID de l'instance si vous publiez depuis Amazon EC2, sinon d'une valeur de remplacement de `/aws/messaging`.

- Vous pouvez annuler cela via `AddMessageSource` ou `AddMessageSourceSuffix` sur `IMessageBusBuilder`.
- `"time"` réglé sur le courant `DateTime` en UTC. Cela peut être contourné en implémentant le vôtre `IDataTimeHandler` et en l'injectant dans le conteneur DI.
- `"data"` contient une représentation JSON de l'objet .NET envoyé ou reçu sous forme de message :
 - `ConfigureSerializationOptions` sur `MessageBusBuilder` permet de configurer le [System.Text.Json.JsonSerializerOptions](#) qui sera utilisé lors de la sérialisation et de la désérialisation du message.
 - Pour injecter des attributs supplémentaires ou transformer l'enveloppe du message une fois que le framework l'a créée, vous pouvez l'implémenter `ISerializationCallback` et l'enregistrer via `AddSerializationCallback` on `MessageBusBuilder`.

Sécurité de l'infrastructure de traitement des AWS messages pour .NET

Le cadre de traitement des AWS messages pour .NET repose sur le AWS SDK pour .NET pour communiquer avec AWS. Pour plus d'informations sur la sécurité dans le AWS SDK pour .NET, voir [Sécurité de ce AWS produit ou service](#).

Pour des raisons de sécurité, le framework n'enregistre pas les messages de données envoyés par l'utilisateur. Si vous souhaitez activer cette fonctionnalité à des fins de débogage, vous devez appeler `EnableDataMessageLogging()` le bus de messages comme suit :

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Si vous découvrez un problème de sécurité potentiel, reportez-vous à la [politique de sécurité](#) pour obtenir des informations de rapport.

Intégration AWS à .NET Aspire dans le AWS SDK pour .NET

.NET Aspire est une nouvelle façon de créer des applications prêtes pour le cloud. En particulier, il fournit une orchestration pour les environnements locaux dans lesquels exécuter, connecter et déboguer les composants des applications distribuées. Pour améliorer la boucle de développement

interne des applications prêtes pour le cloud, des intégrations avec .NET Aspire ont été créées pour connecter vos applications .NET aux AWS ressources. Ces intégrations sont disponibles via le package [NuGet Aspire.Hosting.aws](#).

Les intégrations .NET Aspire suivantes sont disponibles :

- La capacité de fournir vos AWS ressources par le biais de [CloudFormation](#). Cette intégration est utilisée dans le AppHost projet .NET Aspire.

Pour plus d'informations, consultez le billet de blog [Integrating AWS with .NET Aspire](#).

- Installation, configuration et connexion du AWS SDK pour .NET à [Amazon DynamoDB local](#). Cette intégration est utilisée dans le AppHost projet .NET Aspire.

Pour plus d'informations, consultez le billet de blog [Integrating AWS with .NET Aspire](#).

- Activez un environnement de développement local pour les [AWS Lambda](#) fonctions. Cette intégration est utilisée dans le AppHost projet .NET Aspire.

Pour plus d'informations, consultez les articles de blog [Création et débogage d'applications .NET Lambda avec .NET Aspire \(partie 1\)](#) et [Création et débogage d'applications .NET Lambda avec .NET Aspire \(partie 2\)](#).

Note

Ceci est une documentation préliminaire pour une fonctionnalité en version de prévisualisation. Elle est susceptible d'être modifiée.

Comme cette fonctionnalité est en version préliminaire, vous devrez vous inscrire pour les fonctionnalités de prévisualisation. Pour plus d'informations sur cette fonctionnalité de prévisualisation et sur la procédure à suivre pour s'y inscrire, consultez le [problème du suivi du développement sur GitHub](#).

Informations supplémentaires

Pour plus d'informations et de détails sur la façon dont vous pouvez utiliser les intégrations disponibles dans [Aspire.Hosting.aws](#), consultez les ressources suivantes.

- Article de blog [Intégration AWS avec .NET Aspire](#).

- Articles de blog [Création et débogage d'applications .NET Lambda avec .NET Aspire \(partie 1\)](#) et [Création et débogage d'applications .NET Lambda avec .NET Aspire \(partie 2\)](#).
- Le dépôt [integrations-on-dotnet-aspire-for-aws est activé](#). GitHub
- Le [fichier README](#) détaillé pour le package [NuGet Aspire.Hosting.aws](#).

Support pour d'autres AWS services et configurations

Les AWS services de AWS SDK pour .NET soutien s'ajoutent à ceux décrits dans les sections précédentes. Pour plus d'informations sur tous les services pris en charge, consultez la [référence des AWS SDK pour .NET API](#). APIs

Outre les espaces de noms pour les AWS services individuels, il fournit AWS SDK pour .NET également les éléments suivants : APIs

Area	Description	Ressources
AWS Support	Accès programmatique aux dossiers de AWS Support et aux fonctionnalités de Trusted Advisor.	Voir Amazon. AWSSupport et Amazon. AWSSupport.Util .
Général	Énumérations et classes d'assistance.	Consultez Amazon et Amazon.Util .

SDK pour .NET exemples de code

Les exemples de code présentés dans cette rubrique vous montrent comment utiliser le AWS SDK pour .NET with AWS.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Certains services contiennent des exemples de catégories supplémentaires qui montrent comment tirer parti des bibliothèques ou des fonctions spécifiques au service.

Services

- [Exemples d'ACM utilisant SDK pour .NET](#)
- [Exemples d'API Gateway utilisant SDK pour .NET](#)
- [Exemples d'utilisation d'Aurora SDK pour .NET](#)
- [Exemples d'Auto Scaling utilisant SDK pour .NET](#)
- [Exemples d'Amazon Bedrock Runtime utilisant SDK pour .NET](#)
- [CloudWatch Exemples de journaux utilisant SDK pour .NET](#)
- [Exemples d'utilisation du fournisseur d'identité Amazon Cognito SDK pour .NET](#)
- [Exemples d'Amazon Comprehend utilisant SDK pour .NET](#)
- [Exemples d'Amazon DocumentDB utilisant SDK pour .NET](#)
- [Exemples DynamoDB utilisant SDK pour .NET](#)
- [EC2 Exemples d'Amazon utilisant SDK pour .NET](#)
- [Exemples d'Amazon ECS utilisant SDK pour .NET](#)
- [Elastic Load Balancing - Version 2 : exemples d'utilisation SDK pour .NET](#)
- [EventBridge exemples utilisant SDK pour .NET](#)
- [EventBridge Exemples de planificateur utilisant SDK pour .NET](#)

- [Exemples d'utilisation d'Amazon Glacier SDK pour .NET](#)
- [AWS Glue exemples utilisant SDK pour .NET](#)
- [Exemples d'IAM utilisant SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon Keyspaces SDK pour .NET](#)
- [Exemples d'utilisation de Kinesis SDK pour .NET](#)
- [AWS KMS exemples utilisant SDK pour .NET](#)
- [Exemples Lambda utilisant SDK pour .NET](#)
- [MediaConvert exemples utilisant SDK pour .NET](#)
- [Exemples d'Amazon MSK utilisant SDK pour .NET](#)
- [Organisations utilisant des exemples SDK pour .NET](#)
- [Exemples d'utilisation de Partner Central SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon Pinpoint SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon Polly SDK pour .NET](#)
- [Exemples d'Amazon RDS utilisant SDK pour .NET](#)
- [Exemples d'Amazon RDS Data Service utilisant SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon Rekognition SDK pour .NET](#)
- [Exemples d'enregistrement de domaine Route 53 à l'aide de SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon S3 SDK pour .NET](#)
- [SageMaker Exemples d'IA utilisant SDK pour .NET](#)
- [Exemples d'utilisation de Secrets Manager SDK pour .NET](#)
- [Exemples d'Amazon SES utilisant SDK pour .NET](#)
- [Exemples d'API Amazon SES v2 utilisant SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon SNS SDK pour .NET](#)
- [Exemples d'utilisation d'Amazon SQS SDK pour .NET](#)
- [Exemples de Step Functions utilisant SDK pour .NET](#)
- [AWS STS exemples utilisant SDK pour .NET](#)
- [Support exemples utilisant SDK pour .NET](#)
- [Exemples d'Amazon Textract utilisant SDK pour .NET](#)
- [Exemples d'Amazon Transcribe utilisant SDK pour .NET](#)
- [Exemples d'Amazon Translate utilisant SDK pour .NET](#)

Exemples d'ACM utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with ACM.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

DescribeCertificate

L'exemple de code suivant montre comment utiliser `DescribeCertificate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.CertificateManager;
using Amazon.CertificateManager.Model;

namespace DescribeCertificate
{
    class DescribeCertificate
    {
        // The following example retrieves and displays the metadata for a
```

```

// certificate using the AWS Certificate Manager (ACM) service.

// Specify your AWS Region (an example Region is shown).
private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
private static AmazonCertificateManagerClient _client;

static void Main(string[] args)
{
    _client = new
Amazon.CertificateManager.AmazonCertificateManagerClient(ACMRegion);

    var describeCertificateReq = new DescribeCertificateRequest();
    // The ARN used here is just an example. Replace it with the ARN of
    // a certificate that exists on your account.
    describeCertificateReq.CertificateArn =
        "arn:aws:acm:us-
east-1:123456789012:certificate/8cfd7dae-9b6a-2d07-92bc-1c309EXAMPLE";

    var certificateDetailResp =
        DescribeCertificateResponseAsync(client: _client, request:
describeCertificateReq);
    var certificateDetail = certificateDetailResp.Result.Certificate;

    if (certificateDetail is not null)
    {
        DisplayCertificateDetails(certificateDetail);
    }
}

/// <summary>
/// Displays detailed metadata about a certificate retrieved
/// using the ACM service.
/// </summary>
/// <param name="certificateDetail">The object that contains details
/// returned from the call to DescribeCertificateAsync.</param>
static void DisplayCertificateDetails(CertificateDetail certificateDetail)
{
    Console.WriteLine("\nCertificate Details: ");
    Console.WriteLine($"Certificate Domain:
{certificateDetail.DomainName}");
    Console.WriteLine($"Certificate Arn:
{certificateDetail.CertificateArn}");
    Console.WriteLine($"Certificate Subject: {certificateDetail.Subject}");
    Console.WriteLine($"Certificate Status: {certificateDetail.Status}");
}

```

```

        foreach (var san in certificateDetail.SubjectAlternativeNames)
        {
            Console.WriteLine($"Certificate SubjectAlternativeName: {san}");
        }
    }

    /// <summary>
    /// Retrieves the metadata associated with the ACM service certificate.
    /// </summary>
    /// <param name="client">An AmazonCertificateManagerClient object
    /// used to call DescribeCertificateResponse.</param>
    /// <param name="request">The DescribeCertificateRequest object that
    /// will be passed to the method call.</param>
    /// <returns></returns>
    static async Task<DescribeCertificateResponse>
DescribeCertificateResponseAsync(
    AmazonCertificateManagerClient client, DescribeCertificateRequest
request)
    {
        var response = new DescribeCertificateResponse();

        try
        {
            response = await client.DescribeCertificateAsync(request);
        }
        catch (InvalidArnException)
        {
            Console.WriteLine($"Error: The ARN specified is invalid.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Error: The specified certificate could not be
found.");
        }

        return response;
    }
}
}

```


- Pour plus de détails sur l'API, reportez-vous [DescribeCertificate](#) à la section Référence des AWS SDK pour .NET API.

ListCertificates

L'exemple de code suivant montre comment utiliser `ListCertificates`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.CertificateManager;
using Amazon.CertificateManager.Model;

namespace ListCertificates
{
    // The following example retrieves and displays a list of the
    // certificates defined for the default account using the AWS
    // Certificate Manager (ACM) service.
    class ListCertificates
    {
        // Specify your AWS Region (an example Region is shown).

        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new AmazonCertificateManagerClient(ACMRegion);
            var certificateList = ListCertificatesResponseAsync(client: _client);

            Console.WriteLine("Certificate Summary List\n");
        }
    }
}
```

```

        foreach (var certificate in
certificateList.Result.CertificateSummaryList)
        {
            Console.WriteLine($"Certificate Domain: {certificate.DomainName}");
            Console.WriteLine($"Certificate ARN:
{certificate.CertificateArn}\n");
        }
    }

    /// <summary>
    /// Retrieves a list of the certificates defined in this Region.
    /// </summary>
    /// <param name="client">The ACM client object passed to the
    /// ListCertificateResAsync method call.</param>
    /// <param name="request"></param>
    /// <returns>The ListCertificatesResponse.</returns>
    static async Task<ListCertificatesResponse> ListCertificatesResponseAsync(
        AmazonCertificateManagerClient client)
    {
        var request = new ListCertificatesRequest();

        var response = await client.ListCertificatesAsync(request);
        return response;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListCertificates](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'API Gateway utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with API Gateway.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

AWS les contributions communautaires sont des exemples qui ont été créés et sont maintenus par plusieurs équipes AWS. Pour faire un commentaire, utilisez le mécanisme fourni dans les référentiels liés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Scénarios](#)
- [AWS contributions communautaires](#)

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS contributions communautaires

Création et test d'une application sans serveur

L'exemple de code suivant montre comment créer et tester une application sans serveur à l'aide d'API Gateway avec Lambda et DynamoDB.

SDK pour .NET

Illustre comment créer et tester une application sans serveur composée d'une API Gateway avec Lambda et DynamoDB à l'aide du kit SDK .NET.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

Exemples d'utilisation d'Aurora SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Aurora.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Scénarios](#)

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser le AWS SDK pour .NET pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front-end créé avec React.js pour interagir avec un backend RESTful .NET.

- Intégrez une application Web React à AWS des services.
- Listez, ajoutez et mettez à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés à l'aide d'Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples d'Auto Scaling utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with Auto Scaling.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

AttachLoadBalancerTargetGroups

L'exemple de code suivant montre comment utiliser `AttachLoadBalancerTargetGroups`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
/// <returns>Async task.</returns>
public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
{
    await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
        new AttachLoadBalancerTargetGroupsRequest()
        {
```

```

        AutoScalingGroupName = autoScalingGroupName,
        TargetGroupARNs = new List<string>() { targetGroupArn }
    });
}

```

- Pour plus de détails sur l'API, reportez-vous [AttachLoadBalancerTargetGroups](#) à la section Référence des AWS SDK pour .NET API.

DeleteAutoScalingGroup

L'exemple de code suivant montre comment utiliser `DeleteAutoScalingGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Mettez à jour la taille minimale d'un groupe Auto Scaling à zéro, mettez fin à toutes les instances du groupe et supprimez le groupe.

```

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)
    {
        try
        {
            await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                new TerminateInstanceInAutoScalingGroupRequest()
                {
                    InstanceId = instanceId,
                    ShouldDecrementDesiredCapacity = false
                }
            );
        }
        catch { }
    }
}

```

```

        });
        stopping = true;
    }
    catch (ScalingActivityInProgressException)
    {
        Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
        Thread.Sleep(10000);
    }
}

/// <summary>
/// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
/// waits and retries until the group is successfully deleted.
/// </summary>
/// <param name="groupName">The name of the group to try to delete.</param>
/// <returns>Async task.</returns>
public async Task TryDeleteGroupByName(string groupName)
{
    var stopped = false;
    while (!stopped)
    {
        try
        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                });
            stopped = true;
        }
        catch (Exception e)
            when ((e is ScalingActivityInProgressException)
                || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>

```



```

/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
    else
    {
        Console.WriteLine($"No groups found with name {groupName}.");
    }
}

```

```

/// <summary>
/// Delete an Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>

```

```
public async Task<bool> DeleteAutoScalingGroupAsync(
    string groupName)
{
    var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        ForceDelete = true,
    };

    var response = await
        _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"You successfully deleted {groupName}");
        return true;
    }

    Console.WriteLine($"Couldn't delete {groupName}.");
    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteAutoScalingGroup](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.

- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    // Set up dependency injection for the AWS services.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonDynamoDB>()
                .AddAWSService<IAmazonElasticLoadBalancingV2>()
                .AddAWSService<IAmazonSimpleSystemsManagement>()
                .AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalerWrapper>()
```

```

        .AddTransient<ElasticLoadBalancerWrapper>()
        .AddTransient<SmParameterWrapper>()
        .AddTransient<Recommendations>()
        .AddSingleton<IConfiguration>(_configuration)
    )
    .Build();

ServicesSetup(host);
ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
    await DestroyResources(true);
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>

```

```

    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(

```

```

        "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\n\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"

```

```

        + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
        + "that control the flow of the demo.");

    var startupScriptPath = Path.Join(_configuration["resourcePath"],
        "server_startup_script.sh");
    var instancePolicyPath = Path.Join(_configuration["resourcePath"],
        "instance_policy.json");
    await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(
        "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
        + "Availability Zone.\n");
    var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
    await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(
        "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
        + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you're ready to continue.");
    if (interactive)
        Console.ReadLine();

    Console.WriteLine("Creating variables that control the flow of the demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine(
        "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
        + "defines how the load balancer connects to instances. The load
balancer provides a\n"
        + "single endpoint where clients connect and dispatches requests to
instances in the group.");

    var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();

```

```

        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupM
protocol, port, defaultVpc.VpcId);

        await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.Lo
subnetIds, targetGroup);
        await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
_elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
_autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
_autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
default VPC must\n"
                    + "allows access from this computer. You can either add it
automatically from this\n"

```



```

        + "example or add it yourself using the AWS Management Console.
\n");

        if (!interactive || GetYesNoResponse(
            "Do you want to add a rule to the security group to allow
inbound traffic from your computer's IP address?"))
        {
            await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
        }

        if (!sshPortIsOpen)
        {
            if (!interactive || GetYesNoResponse(
                "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
            {
                await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
            }
        }

        loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
        Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
        Console.WriteLine($"http://{endPoint}\n");
    }
    else
    {
        Console.WriteLine(
            "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
            + "manually verifying that your VPC and security group are
configured correctly and that\n"
            + "you can successfully make a GET request to the load balancer
endpoint:\n");
        Console.WriteLine($"http://{endPoint}\n");
    }
}

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
            "to create situations where the web service fails, and
shows how using a resilient\n" +
            "architecture can keep the web service running in spite of
these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
            $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
            $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
        await
            _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    }

```

```

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
            "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);

```

```

        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"Now, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");

```

```
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"Even while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"When all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();
```

```

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
            _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
            _recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }
    }

```

```

        });
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Créez une classe qui englobe les EC2 actions Auto Scaling et Amazon.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.

```

```

    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration,
        ILogger<AutoScalerWrapper> logger)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;
        _logger = logger;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
    specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance. The role has attached policies that specify the AWS permissions
    granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>

```



```

    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\", \" +
                                "\"Statement\": [{\" +
                                    "\"Effect\": \"Allow\", \" +
                                    "\"Principal\": {\" +
                                    "\"Service\": [\" +
                                        \"ec2.amazonaws.com\"\" +
                                    \"]\" +
                                    \",\" +
                                "\"Action\": \"sts:AssumeRole\"\" +
                                \"]}\" +
                                "};

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(

```

```

        new ListPoliciesRequest()
        {
            Scope = PolicyScopeType.Local
        });
    // Get the entire list using the paginator.
    await foreach (var policy in policiesPaginator.Policies)
    {
        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
}

```

```

        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {

```

```

        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
    after
    /// the instance is started. This script installs the Python packages and starts
    a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
    run.</param>

```

```

    /// <param name="instancePolicyPath">The path to a permissions policy to create
    and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);
            await CreateInstanceProfileWithName(_instancePolicyName,
_instanceRoleName,
            _instanceProfileName, instancePolicyPath);

            var startServerText = await File.ReadAllTextAsync(startupScriptPath);
            var plainTextBytes =
System.Text.Encoding.UTF8.GetBytes(startServerText);

            var amiLatest = await _amazonSsm.GetParameterAsync(
                new GetParameterRequest() { Name = _amiParam });
            var amiId = amiLatest.Parameter.Value;
            var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
                new CreateLaunchTemplateRequest()
                {
                    LaunchTemplateName = _launchTemplateName,
                    LaunchTemplateData = new RequestLaunchTemplateData()
                    {
                        InstanceType = _instanceType,
                        ImageId = amiId,
                        IamInstanceProfile =
                            new
LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                        KeyName = _keyPairName,
                        UserData = System.Convert.ToBase64String(plainTextBytes)
                    }
                });
            return launchTemplateResponse.LaunchTemplate;
        }
        catch (AmazonEC2Exception ec2Exception)
        {

```

```

        if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.AlreadyExistsException")
        {
            _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                $"Please try again with a unique name.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    try
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while listing availability zones.:
{ex.Message}");
        throw;
    }
}

```

```

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
    List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        try

```

```

        {
            var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
                new DescribeVpcsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("is-default", new List<string>() { "true" })
                    }
                });
            return vpcResponse.Vpcs[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "UnauthorizedOperation")
            {
                _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        try
        {
            var subnets = new List<Subnet>();
            var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
                new DescribeSubnetsRequest()
                {

```



```

        Filters = new List<Amazon.EC2.Model.Filter>()
        {
            new("vpc-id", new List<string>() { vpcId }),
            new("availability-zone", availabilityZones),
            new("default-for-az", new List<string>() { "true" })
        }
    });

    // Get the entire list using the paginator.
    await foreach (var subnet in subnetPaginator.Subnets)
    {
        subnets.Add(subnet);
    }

    return subnets;
}
catch (AmazonEC2Exception ec2Exception)
{
    if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
    {
        _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId} does
not exist.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while describing the subnets.:
{ex.Message}");
    throw;
}
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(

```

```

        new DeleteLaunchTemplateRequest()
        {
            LaunchTemplateName = templateName
        });
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
            "InvalidLaunchTemplateName.NotFoundException")
        {
            _logger.LogError(
                $"Could not delete the template, the name {_launchTemplateName}
was not found.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(

```

```

        new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                });
            // Delete the custom policies only.
            if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
            {
                await _amazonIam.DeletePolicyAsync(
                    new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                    {
                        PolicyArn = policy.PolicyArn
                    });
            }
        }

        await _amazonIam.DeleteRoleAsync(
            new DeleteRoleRequest() { RoleName = roleName });
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine($"Instance profile {profileName} does not exist.");
    }
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
        new DescribeAutoScalingGroupsRequest()
        {

```

```

        AutoScalingGroupNames = new List<string>() { group }
    });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    try
    {
        var response = await
        _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while creating the template.:
{ex.Message}");
        throw;
    }
}
}

```

```

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
    replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
    ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
    the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
    the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
    credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
                    instancesPaginator.InstanceInformationList)

```

```

        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine("Waiting for instance to be running.");
    await WaitForInstanceState(instanceId, InstanceStateName.Running);
    Console.WriteLine("Instance ready.");
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
            DocumentName = "AWS-RunShellScript",
            Parameters = new Dictionary<string, List<string>>()
            {
                {
                    "commands",
                    new List<string>() { "cd / && sudo python3 server.py"
80" }
                }
            }
        });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}
catch (AmazonEC2Exception ec2Exception)
{
    if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
    {
        _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while replacing the template.:
{ex.Message}");
    throw;
}
}

```

```

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
    progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try

```

```

        {
            await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                new DeleteAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName
                });
            stopped = true;
        }
        catch (Exception e)
            when ((e is ScalingActivityInProgressException)
                || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }
    }
}

```



```

    }

    await TryDeleteGroupByName(groupName);
}
else
{
    Console.WriteLine($"No groups found with name {groupName}.");
}
}

/// <summary>
/// Get the default security group for a specified Vpc.
/// </summary>
/// <param name="vpc">The Vpc to search.</param>
/// <returns>The default security group.</returns>
public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
{
    var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
        new DescribeSecurityGroupsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("group-name", new List<string>() { "default" }),
                new ("vpc-id", new List<string>() { vpc.VpcId })
            }
        });
    return groupResponse.SecurityGroups[0];
}

/// <summary>
/// Verify the default security group of a Vpc allows ingress from the calling
computer.
/// This can be done by allowing ingress from this computer's IP address.
/// In some situations, such as connecting from a corporate network, you must
instead specify
/// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
/// If you do, be sure to remove public access when you're done.
/// </summary>
/// <param name="vpc">The group to check.</param>
/// <param name="port">The port to verify.</param>
/// <param name="ipAddress">This computer's IP address.</param>
/// <returns>True if the ip address is allowed on the group.</returns>

```

```

    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }

                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>

```

```

    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
    Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
    param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
    targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }

```

```

    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
        InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is in the specified state.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEc2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.WriteLine(". ");
        } while (!hasState);

        return hasState;
    }
}

```

Créez une classe qui englobe les actions Elastic Load Balancing.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{

```

```

private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
private string? _endpoint = null;
private readonly string _targetGroupName = "";
private readonly string _loadBalancerName = "";
HttpClient _httpClient = new();

public string TargetGroupName => _targetGroupName;
public string LoadBalancerName => _loadBalancerName;

/// <summary>
/// Constructor for the Elastic Load Balancer wrapper.
/// </summary>
/// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
/// <param name="configuration">The injected configuration.</param>
public ElasticLoadBalancerWrapper(
    IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
    IConfiguration configuration)
{
    _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
    var prefix = configuration["resourcePrefix"];
    _targetGroupName = prefix + "-tg";
    _loadBalancerName = prefix + "-lb";
}

/// <summary>
/// Get the HTTP Endpoint of a load balancer by its name.
/// </summary>
/// <param name="loadBalancerName">The name of the load balancer.</param>
/// <returns>The HTTP endpoint.</returns>
public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
{
    if (_endpoint == null)
    {
        var endpointResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { loadBalancerName }
                });
        _endpoint = endpointResponse.LoadBalancers[0].DNSName;
    }
}

```

```

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
    CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {

```

```

        Console.WriteLine($"Target group {groupName} not found.");
    }
    return result;
}

/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
/// <returns>The new TargetGroup object.</returns>
public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
{
    var createResponse = await
_amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
    new CreateTargetGroupRequest()
    {
        Name = groupName,
        Protocol = protocol,
        Port = port,
        HealthCheckPath = "/healthcheck",
        HealthCheckIntervalSeconds = 10,
        HealthCheckTimeoutSeconds = 5,
        HealthyThresholdCount = 2,
        UnhealthyThresholdCount = 2,
        VpcId = vpcId
    });
    var targetGroup = createResponse.TargetGroups[0];
    return targetGroup;
}

/// <summary>

```

```

    /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
    }

```



```

    }
    // Create the listener.
    await _amazonElasticLoadBalancingV2.CreateListenerAsync(
        new CreateListenerRequest()
        {
            LoadBalancerArn = loadBalancerArn,
            Protocol = targetGroup.Protocol,
            Port = targetGroup.Port,
            DefaultActions = new List<Action>()
            {
                new Action()
                {
                    Type = ActionTypeEnum.Forward,
                    TargetGroupArn = targetGroup.TargetGroupArn
                }
            }
        });
    return createLbResponse.LoadBalancers[0];
}

/// <summary>
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {

```

```

        retries = 0;
    }
}
catch (HttpRequestException)
{
    Console.WriteLine("Connection error, retrying...");
    retries--;
    Thread.Sleep(10000);
}
}

return success;
}

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
    try
    {
        var describeLoadBalancerResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });
        var lbArn =
            describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
        await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
            new DeleteLoadBalancerRequest()
            {
                LoadBalancerArn = lbArn
            }
        );
    }
    catch (LoadBalancerNotFoundException)
    {
        Console.WriteLine($"Load balancer {name} not found.");
    }
}

```

```

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}

```

Créez une classe qui utilise DynamoDB pour simuler un service de recommandation.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.WriteLine($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition()
                    {
                        AttributeName = "MediaType",

```

```

        AttributeType = ScalarAttributeType.S
    },
    new AttributeDefinition()
    {
        AttributeName = "ItemId",
        AttributeType = ScalarAttributeType.N
    }
},
KeySchema = new List<KeySchemaElement>()
{
    new KeySchemaElement()
    {
        AttributeName = "MediaTypeId",
        KeyType = KeyType.HASH
    },
    new KeySchemaElement()
    {
        AttributeName = "ItemId",
        KeyType = KeyType.RANGE
    }
},
ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = 5,
    WriteCapacityUnits = 5
}
};

await _amazonDynamoDb.CreateTableAsync(createRequest);

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("\nWaiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = tableName
};

TableStatus status;
do
{
    Thread.Sleep(2000);

    var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);

```

```

        status = describeTableResponse.Table.TableStatus;

        Console.Write(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
    return false;
}
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)

```

```

    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}

```

Créez une classe qui englobe les actions de Systems Manager.

```

/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
/// parameters
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.

```

```

    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
    }
}

```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)

- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplaceIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Exemples d'Amazon Bedrock Runtime utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Bedrock Runtime.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres

Services AWS

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Scénarios](#)
- [Amazon Nova](#)
- [Amazon Nova Canvas](#)
- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta Llama](#)
- [Mistral AI](#)

Scénarios

Création d'une application de terrain de jeu pour interagir avec les modèles de fondation Amazon Bedrock

L'exemple de code suivant montre comment créer des terrains de jeu pour interagir avec les modèles de fondation Amazon Bedrock selon différentes modalités.

SDK pour .NET

Le terrain de jeu du modèle de fondation (FM) .NET est un exemple d'application .NET MAUI Blazor qui montre comment utiliser Amazon Bedrock à partir du code C#. Cet exemple montre comment les développeurs .NET et C# peuvent utiliser Amazon Bedrock pour créer des applications basées sur l'IA générative. Vous pouvez tester les modèles de fondation Amazon Bedrock et interagir avec en utilisant les quatre terrains de jeu suivants :

- un terrain de jeu de texte ;
- un terrain de jeu de chat ;
- un terrain de jeu de chat vocal ;
- un terrain de jeu d'images.

L'exemple répertorie et affiche également les modèles de fondation auxquels vous avez accès ainsi que leurs caractéristiques. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Bedrock Runtime

Utilisation de l'outil avec l'API Converse

L'exemple de code suivant montre comment créer une interaction typique entre une application, un modèle d'IA génératif et des outils connectés ou comment APIs arbitrer les interactions entre l'IA et le monde extérieur. Il utilise comme exemple la connexion d'une API de météorologie externe au modèle d'IA afin de fournir des informations météorologiques en temps réel en fonction des données saisies par l'utilisateur.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécution principale du flux de scénario. Ce scénario orchestre la conversation entre l'utilisateur, l'API Converse Amazon Bedrock et un outil météo.

```
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using Amazon.Runtime.Documents;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Http;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;

namespace ConverseToolScenario;

public static class ConverseToolScenario
{
    /*
     Before running this .NET code example, set up your development environment,
     including your credentials.
```

This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a weather tool.

The script interacts with a foundation model on Amazon Bedrock to provide weather information based on user

input. It uses the Open-Meteo API (<https://open-meteo.com>) to retrieve current weather data for a given location.

```
*/
```

```
public static BedrockActionsWrapper _bedrockActionsWrapper = null!;  
public static WeatherTool _weatherTool = null!;  
public static bool _interactive = true;
```

```
// Change this string to use a different model with Converse API.  
private static string model_id = "amazon.nova-lite-v1:0";
```

```
private static string system_prompt = @"
```

```
    You are a weather assistant that provides current weather data for user-  
specified locations using only
```

```
    the Weather_Tool, which expects latitude and longitude. Infer the  
coordinates from the location yourself.
```

```
    If the user specifies a state, country, or region, infer the locations of  
cities within that state.
```

```
    If the user provides coordinates, infer the approximate location and refer  
to it in your response.
```

```
    To use the tool, you strictly apply the provided tool specification.
```

```
    - Explain your step-by-step process, and give brief updates before each  
step.
```

```
    - Only use the Weather_Tool for data. Never guess or make up information.
```

```
    - Repeat the tool use for subsequent requests if necessary.
```

```
    - If the tool errors, apologize, explain weather is unavailable, and suggest  
other options.
```

```
    - Report temperatures in °C (°F) and wind in km/h (mph). Keep weather  
reports concise. Sparingly use  
emojis where appropriate.
```

```
    - Only respond to weather queries. Remind off-topic users of your purpose.
```

```
    - Never claim to search online, access external data, or use tools besides  
Weather_Tool.
```

```
    - Complete the entire process until you have all required data before  
sending the complete response.
```

```
"
```

```
;
```

```

private static string default_prompt = "What is the weather like in Seattle?";

// The maximum number of recursive calls allowed in the tool use function.
// This helps prevent infinite loops and potential performance issues.
private static int max_recurSIONS = 5;

public static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Error)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddHttpClient()
                .AddSingleton<IAmazonBedrockRuntime>(_ => new
AmazonBedrockRuntimeClient(RegionEndpoint.USEast1)) // Specify a region that has
access to the chosen model.
                .AddTransient<BedrockActionsWrapper>()
                .AddTransient<WeatherTool>()
                .RemoveAll<IHttpMessageHandlerBuilderFilter>()
            )
        .Build();

    ServicesSetup(host);

    try
    {
        await RunConversationAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        Console.WriteLine(new string('-', 80));
    }
    finally
    {
        Console.WriteLine(
            "Amazon Bedrock Converse API with Tool Use Feature Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
}

```

```

    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _bedrockActionsWrapper =
host.Services.GetRequiredService<BedrockActionsWrapper>();
    _weatherTool = host.Services.GetRequiredService<WeatherTool>();
}

/// <summary>
/// Starts the conversation with the user and handles the interaction with
Bedrock.
/// </summary>
/// <returns>The conversation array.</returns>
public static async Task<List<Message>> RunConversationAsync()
{
    // Print the greeting and a short user guide
    PrintHeader();

    // Start with an empty conversation
    var conversation = new List<Message>();

    // Get the first user input
    var userInput = await GetUserInputAsync();

    while (userInput != null)
    {
        // Create a new message with the user input and append it to the
conversation
        var message = new Message { Role = ConversationRole.User, Content = new
List<ContentBlock> { new ContentBlock { Text = userInput } } };
        conversation.Add(message);

        // Send the conversation to Amazon Bedrock
        var bedrockResponse = await SendConversationToBedrock(conversation);

        // Recursively handle the model's response until the model has returned
its final response or the recursion counter has reached 0
    }
}

```

```

        await ProcessModelResponseAsync(bedrockResponse, conversation,
max_recurSIONs);

        // Repeat the loop until the user decides to exit the application
        userInput = await GetUserInputAsync();
    }

    PrintFooter();
    return conversation;
}

/// <summary>
/// Sends the conversation, the system prompt, and the tool spec to Amazon
Bedrock, and returns the response.
/// </summary>
/// <param name="conversation">The conversation history including the next
message to send.</param>
/// <returns>The response from Amazon Bedrock.</returns>
private static async Task<ConverseResponse>
SendConversationToBedrock(List<Message> conversation)
{
    Console.WriteLine("\tCalling Bedrock...");

    // Send the conversation, system prompt, and tool configuration, and return
the response
    return await _bedrockActionsWrapper.SendConverseRequestAsync(model_id,
system_prompt, conversation, _weatherTool.GetToolSpec());
}

/// <summary>
/// Processes the response received via Amazon Bedrock and performs the
necessary actions based on the stop reason.
/// </summary>
/// <param name="modelResponse">The model's response returned via Amazon
Bedrock.</param>
/// <param name="conversation">The conversation history.</param>
/// <param name="maxRecursion">The maximum number of recursive calls allowed.</
param>
private static async Task ProcessModelResponseAsync(ConverseResponse
modelResponse, List<Message> conversation, int maxRecursion)
{
    if (maxRecursion <= 0)
    {

```

```

        // Stop the process, the number of recursive calls could indicate an
infinite loop
        Console.WriteLine("\tWarning: Maximum number of recursions reached.
Please try again.");
    }

    // Append the model's response to the ongoing conversation
conversation.Add(modelResponse.Output.Message);

    if (modelResponse.StopReason == "tool_use")
    {
        // If the stop reason is "tool_use", forward everything to the tool use
handler
        await HandleToolUseAsync(modelResponse.Output, conversation,
maxRecursion - 1);
    }

    if (modelResponse.StopReason == "end_turn")
    {
        // If the stop reason is "end_turn", print the model's response text,
and finish the process
        PrintModelResponse(modelResponse.Output.Message.Content[0].Text);
        if (!_interactive)
        {
            default_prompt = "x";
        }
    }
}

/// <summary>
/// Handles the tool use case by invoking the specified tool and sending the
tool's response back to Bedrock.
/// The tool response is appended to the conversation, and the conversation is
sent back to Amazon Bedrock for further processing.
/// </summary>
/// <param name="modelResponse">The model's response containing the tool use
request.</param>
/// <param name="conversation">The conversation history.</param>
/// <param name="maxRecursion">The maximum number of recursive calls allowed.</
param>
public static async Task HandleToolUseAsync(ConverseOutput modelResponse,
List<Message> conversation, int maxRecursion)
{
    // Initialize an empty list of tool results

```



```

var toolResults = new List<ContentBlock>();

// The model's response can consist of multiple content blocks
foreach (var contentBlock in modelResponse.Message.Content)
{
    if (!String.IsNullOrEmpty(contentBlock.Text))
    {
        // If the content block contains text, print it to the console
        PrintModelResponse(contentBlock.Text);
    }

    if (contentBlock.ToolUse != null)
    {
        // If the content block is a tool use request, forward it to the
        tool
        var toolResponse = await InvokeTool(contentBlock.ToolUse);

        // Add the tool use ID and the tool's response to the list of
        results
        toolResults.Add(new ContentBlock
        {
            ToolResult = new ToolResultBlock()
            {
                ToolUseId = toolResponse.ToolUseId,
                Content = new List<ToolResultContentBlock>()
                { new ToolResultContentBlock { Json =
toolResponse.Content } }
            }
        });
    }
}

// Embed the tool results in a new user message
var message = new Message() { Role = ConversationRole.User, Content =
toolResults };

// Append the new message to the ongoing conversation
conversation.Add(message);

// Send the conversation to Amazon Bedrock
var response = await SendConversationToBedrock(conversation);

// Recursively handle the model's response until the model has returned its
final response or the recursion counter has reached 0

```

```

        await ProcessModelResponseAsync(response, conversation, maxRecursion);
    }

    /// <summary>
    /// Invokes the specified tool with the given payload and returns the tool's
response.
    /// If the requested tool does not exist, an error message is returned.
    /// </summary>
    /// <param name="payload">The payload containing the tool name and input data.</
param>
    /// <returns>The tool's response or an error message.</returns>
    public static async Task<ToolResponse> InvokeTool(ToolUseBlock payload)
    {
        var toolName = payload.Name;

        if (toolName == "Weather_Tool")
        {
            var inputData = payload.Input.AsDictionary();
            PrintToolUse(toolName, inputData);

            // Invoke the weather tool with the input data provided
            var weatherResponse = await
_weatherTool.FetchWeatherDataAsync(inputData["latitude"].ToString(),
inputData["longitude"].ToString());
            return new ToolResponse { ToolUseId = payload.ToolUseId, Content =
weatherResponse };
        }
        else
        {
            var errorMessage = $"\\tThe requested tool with name '{toolName}' does
not exist.";
            return new ToolResponse { ToolUseId = payload.ToolUseId, Content = new
{ error = true, message = errorMessage } };
        }
    }

    /// <summary>
    /// Prompts the user for input and returns the user's response.
    /// Returns null if the user enters 'x' to exit.
    /// </summary>
    /// <param name="prompt">The prompt to display to the user.</param>
    /// <returns>The user's input or null if the user chooses to exit.</returns>

```

```

private static async Task<string?> GetUserInputAsync(string prompt = "\tYour
weather info request:")
{
    var userInput = default_prompt;
    if (_interactive)
    {
        Console.WriteLine(new string('*', 80));
        Console.WriteLine($"{prompt} (x to exit): \n\t");
        userInput = Console.ReadLine();
    }

    if (string.IsNullOrEmpty(userInput))
    {
        prompt = "\tPlease enter your weather info request, e.g. the name of a
city";
        return await GetUserInputAsync(prompt);
    }

    if (userInput.ToLowerInvariant() == "x")
    {
        return null;
    }

    return userInput;
}

/// <summary>
/// Logs the welcome message and usage guide for the tool use demo.
/// </summary>
public static void PrintHeader()
{
    Console.WriteLine(@"
=====
Welcome to the Amazon Bedrock Tool Use demo!
=====

    This assistant provides current weather information for user-specified
locations.

    You can ask for weather details by providing the location name or
coordinates. Weather information
will be provided using a custom Tool and open-meteo API.

    Example queries:
    - What's the weather like in New York?

```

- Current weather for latitude 40.70, longitude -74.01
- Is it warmer in Rome or Barcelona today?

To exit the program, simply type 'x' and press Enter.

P.S.: You're not limited to single locations, or even to using English!
Have fun and experiment with the app!

");

}

/// <summary>

/// Logs the footer information for the tool use demo.

/// </summary>

public static void PrintFooter()

{

 Console.WriteLine(@"

=====

Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you
learned something new, or got some inspiration for your own apps today!

For more Bedrock examples in different programming languages, have a look
at:

<https://docs.aws.amazon.com/bedrock/latest/userguide/>

service_code_examples.html

=====

");

}

/// <summary>

/// Logs information about the tool use.

/// </summary>

/// <param name="toolName">The name of the tool being used.</param>

/// <param name="inputData">The input data for the tool.</param>

public static void PrintToolUse(string toolName, Dictionary<string, Document>
inputData)

{

 Console.WriteLine(\$"\\n\\tInvoking tool: {toolName} with input:
{inputData["latitude"].ToString()}, {inputData["longitude"].ToString()}...\\n");

}

/// <summary>

/// Logs the model's response.

/// </summary>

/// <param name="message">The model's response message.</param>

```

    public static void PrintModelResponse(string message)
    {
        Console.WriteLine("\tThe model's response:\n");
        Console.WriteLine(message);
        Console.WriteLine();
    }
}

```

Outil météo utilisé par la démonstration. Ce fichier définit les spécifications de l'outil et met en œuvre la logique permettant de récupérer les données météorologiques à l'aide de l'API Open-Meteo.

```

using Amazon.BedrockRuntime.Model;
using Amazon.Runtime.Documents;
using Microsoft.Extensions.Logging;

namespace ConverseToolScenario;

/// <summary>
/// Weather tool that will be invoked when requested by the Bedrock response.
/// </summary>
public class WeatherTool
{
    private readonly ILogger<WeatherTool> _logger;
    private readonly IHttpClientFactory _httpClientFactory;

    public WeatherTool(ILogger<WeatherTool> logger, IHttpClientFactory
httpClientFactory)
    {
        _logger = logger;
        _httpClientFactory = httpClientFactory;
    }

    /// <summary>
    /// Returns the JSON Schema specification for the Weather tool. The tool
specification
    /// defines the input schema and describes the tool's functionality.
    /// For more information, see https://json-schema.org/understanding-json-schema/
reference.
    /// </summary>
    /// <returns>The tool specification for the Weather tool.</returns>

```

```

public ToolSpecification GetToolSpec()
{
    ToolSpecification toolSpecification = new ToolSpecification();

    toolSpecification.Name = "Weather_Tool";
    toolSpecification.Description = "Get the current weather for a given
location, based on its WGS84 coordinates.";

    Document toolSpecDocument = Document.FromObject(
        new
        {
            type = "object",
            properties = new
            {
                latitude = new
                {
                    type = "string",
                    description = "Geographical WGS84 latitude of the location."
                },
                longitude = new
                {
                    type = "string",
                    description = "Geographical WGS84 longitude of the
location."
                }
            },
            required = new[] { "latitude", "longitude" }
        });

    toolSpecification.InputSchema = new ToolInputSchema() { Json =
toolSpecDocument };
    return toolSpecification;
}

/// <summary>
/// Fetches weather data for the given latitude and longitude using the Open-
Meteo API.
/// Returns the weather data or an error message if the request fails.
/// </summary>
/// <param name="latitude">The latitude of the location.</param>
/// <param name="longitude">The longitude of the location.</param>
/// <returns>The weather data or an error message.</returns>
public async Task<Document> FetchWeatherDataAsync(string latitude, string
longitude)

```

```

{
    string endpoint = "https://api.open-meteo.com/v1/forecast";

    try
    {
        var httpClient = _httpClientFactory.CreateClient();
        var response = await httpClient.GetAsync($"{endpoint}?
latitude={latitude}&longitude={longitude}&current_weather=True");
        response.EnsureSuccessStatusCode();
        var weatherData = await response.Content.ReadAsStringAsync();

        Document weatherDocument = Document.FromObject(
            new { weather_data = weatherData });

        return weatherDocument;
    }
    catch (HttpRequestException e)
    {
        _logger.LogError(e, "Error fetching weather data: {Message}",
e.Message);
        throw;
    }
    catch (Exception e)
    {
        _logger.LogError(e, "Unexpected error fetching weather data: {Message}",
e.Message);
        throw;
    }
}
}

```

Action de l'API Converse avec une configuration d'outil.

```

/// <summary>
/// Wrapper class for interacting with the Amazon Bedrock Converse API.
/// </summary>
public class BedrockActionsWrapper
{
    private readonly IAmazonBedrockRuntime _bedrockClient;
    private readonly ILogger<BedrockActionsWrapper> _logger;

```

```

    /// <summary>
    /// Initializes a new instance of the <see cref="BedrockActionsWrapper"/> class.
    /// </summary>
    /// <param name="bedrockClient">The Bedrock Converse API client.</param>
    /// <param name="logger">The logger instance.</param>
    public BedrockActionsWrapper(IAmazonBedrockRuntime bedrockClient,
    ILogger<BedrockActionsWrapper> logger)
    {
        _bedrockClient = bedrockClient;
        _logger = logger;
    }

    /// <summary>
    /// Sends a Converse request to the Amazon Bedrock Converse API.
    /// </summary>
    /// <param name="modelId">The Bedrock Model Id.</param>
    /// <param name="systemPrompt">A system prompt instruction.</param>
    /// <param name="conversation">The array of messages in the conversation.</
param>
    /// <param name="toolSpec">The specification for a tool.</param>
    /// <returns>The response of the model.</returns>
    public async Task<ConverseResponse> SendConverseRequestAsync(string modelId,
    string systemPrompt, List<Message> conversation, ToolSpecification toolSpec)
    {
        try
        {
            var request = new ConverseRequest()
            {
                ModelId = modelId,
                System = new List<SystemContentBlock>()
                {
                    new SystemContentBlock()
                    {
                        Text = systemPrompt
                    }
                },
                Messages = conversation,
                ToolConfig = new ToolConfiguration()
                {
                    Tools = new List<Tool>()
                    {
                        new Tool()
                        {
                            ToolSpec = toolSpec
                        }
                    }
                }
            };

```



```
        }
    }
}

};

var response = await _bedrockClient.ConverseAsync(request);

return response;
}
catch (ModelNotReadyException ex)
{
    _logger.LogError(ex, "Model not ready, please wait and try again.");
    throw;
}
catch (AmazonBedrockRuntimeException ex)
{
    _logger.LogError(ex, "Error occurred while sending Converse request.");
    throw;
}
}
}
```

- Pour plus de détails sur l'API, consultez [Converse](#) dans la Référence des API du kit AWS SDK pour .NET .

Amazon Nova

Converse

L'exemple de code suivant montre comment envoyer un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Envoyez un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock.

```
// Use the Converse API to send a text message to Amazon Nova.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Amazon Nova Lite.
var modelId = "amazon.nova-lite-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
```

```

    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}

```

Envoyez une conversation de messages à Amazon Nova à l'aide de l'API Converse de Bedrock avec une configuration d'outil.

```

/// <summary>
/// Wrapper class for interacting with the Amazon Bedrock Converse API.
/// </summary>
public class BedrockActionsWrapper
{
    private readonly IAmazonBedrockRuntime _bedrockClient;
    private readonly ILogger<BedrockActionsWrapper> _logger;

    /// <summary>
    /// Initializes a new instance of the <see cref="BedrockActionsWrapper"/> class.
    /// </summary>
    /// <param name="bedrockClient">The Bedrock Converse API client.</param>
    /// <param name="logger">The logger instance.</param>
    public BedrockActionsWrapper(IAmazonBedrockRuntime bedrockClient,
    ILogger<BedrockActionsWrapper> logger)
    {
        _bedrockClient = bedrockClient;
        _logger = logger;
    }

    /// <summary>
    /// Sends a Converse request to the Amazon Bedrock Converse API.
    /// </summary>
    /// <param name="modelId">The Bedrock Model Id.</param>
    /// <param name="systemPrompt">A system prompt instruction.</param>

```

```
    /// <param name="conversation">The array of messages in the conversation.</param>
    /// <param name="toolSpec">The specification for a tool.</param>
    /// <returns>The response of the model.</returns>
    public async Task<ConverseResponse> SendConverseRequestAsync(string modelId,
string systemPrompt, List<Message> conversation, ToolSpecification toolSpec)
    {
        try
        {
            var request = new ConverseRequest()
            {
                ModelId = modelId,
                System = new List<SystemContentBlock>()
                {
                    new SystemContentBlock()
                    {
                        Text = systemPrompt
                    }
                },
                Messages = conversation,
                ToolConfig = new ToolConfiguration()
                {
                    Tools = new List<Tool>()
                    {
                        new Tool()
                        {
                            ToolSpec = toolSpec
                        }
                    }
                }
            };

            var response = await _bedrockClient.ConverseAsync(request);

            return response;
        }
        catch (ModelNotReadyException ex)
        {
            _logger.LogError(ex, "Model not ready, please wait and try again.");
            throw;
        }
        catch (AmazonBedrockRuntimeException ex)
        {
            _logger.LogError(ex, "Error occurred while sending Converse request.");
        }
    }
}
```

```
        throw;  
    }  
}  
}
```

- Pour plus de détails sur l'API, consultez [Converse](#) dans la Référence des API du kit AWS SDK pour .NET .

ConverseStream

L'exemple de code suivant montre comment envoyer un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock et comment traiter le flux de réponses en temps réel.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Envoyez un message texte à Amazon Nova à l'aide de l'API Converse de Bedrock et traitez le flux de réponses en temps réel.

```
// Use the Converse API to send a text message to Amazon Nova  
// and print the response stream.  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using Amazon;  
using Amazon.BedrockRuntime;  
using Amazon.BedrockRuntime.Model;  
  
// Create a Bedrock Runtime client in the AWS Region you want to use.  
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);  
  
// Set the model ID, e.g., Amazon Nova Lite.  
var modelId = "amazon.nova-lite-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
userMessage } }
        },
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [ConverseStream](#) à la section Référence des AWS SDK pour .NET API.

Scénario : utilisation de l'outil avec l'API Converse

L'exemple de code suivant montre comment créer une interaction typique entre une application, un modèle d'IA génératif et des outils connectés ou comment APIs arbitrer les interactions entre l'IA et le monde extérieur. Il utilise comme exemple la connexion d'une API de météorologie externe au modèle d'IA afin de fournir des informations météorologiques en temps réel en fonction des données saisies par l'utilisateur.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécution principale du flux de scénario. Ce scénario orchestre la conversation entre l'utilisateur, l'API Converse Amazon Bedrock et un outil météo.

```
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using Amazon.Runtime.Documents;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Http;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;

namespace ConverseToolScenario;

public static class ConverseToolScenario
{

```

```
/*
```

Before running this .NET code example, set up your development environment, including your credentials.

This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a weather tool.

The script interacts with a foundation model on Amazon Bedrock to provide weather information based on user

input. It uses the Open-Meteo API (<https://open-meteo.com>) to retrieve current weather data for a given location.

```
*/
```

```
public static BedrockActionsWrapper _bedrockActionsWrapper = null!;  
public static WeatherTool _weatherTool = null!;  
public static bool _interactive = true;
```

```
// Change this string to use a different model with Converse API.  
private static string model_id = "amazon.nova-lite-v1:0";
```

```
private static string system_prompt = @"
```

You are a weather assistant that provides current weather data for user-specified locations using only

the Weather_Tool, which expects latitude and longitude. Infer the coordinates from the location yourself.

If the user specifies a state, country, or region, infer the locations of cities within that state.

If the user provides coordinates, infer the approximate location and refer to it in your response.

To use the tool, you strictly apply the provided tool specification.

- Explain your step-by-step process, and give brief updates before each step.

- Only use the Weather_Tool for data. Never guess or make up information.

- Repeat the tool use for subsequent requests if necessary.

- If the tool errors, apologize, explain weather is unavailable, and suggest other options.

- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports concise. Sparingly use emojis where appropriate.

- Only respond to weather queries. Remind off-topic users of your purpose.

- Never claim to search online, access external data, or use tools besides Weather_Tool.

- Complete the entire process until you have all required data before sending the complete response.


```

"
;

private static string default_prompt = "What is the weather like in Seattle?";

// The maximum number of recursive calls allowed in the tool use function.
// This helps prevent infinite loops and potential performance issues.
private static int max_recurions = 5;

public static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Error)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddHttpClient()
                .AddSingleton<IAmazonBedrockRuntime>(_ => new
AmazonBedrockRuntimeClient(RegionEndpoint.USEast1)) // Specify a region that has
access to the chosen model.
                .AddTransient<BedrockActionsWrapper>()
                .AddTransient<WeatherTool>()
                .RemoveAll<IHttpMessageHandlerBuilderFilter>()
            )
        .Build();

    ServicesSetup(host);

    try
    {
        await RunConversationAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        Console.WriteLine(new string('-', 80));
    }
    finally
    {
        Console.WriteLine(

```

```

        "Amazon Bedrock Converse API with Tool Use Feature Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _bedrockActionsWrapper =
host.Services.GetRequiredService<BedrockActionsWrapper>();
    _weatherTool = host.Services.GetRequiredService<WeatherTool>();
}

/// <summary>
/// Starts the conversation with the user and handles the interaction with
Bedrock.
/// </summary>
/// <returns>The conversation array.</returns>
public static async Task<List<Message>> RunConversationAsync()
{
    // Print the greeting and a short user guide
    PrintHeader();

    // Start with an empty conversation
    var conversation = new List<Message>();

    // Get the first user input
    var userInput = await GetUserInputAsync();

    while (userInput != null)
    {
        // Create a new message with the user input and append it to the
conversation
        var message = new Message { Role = ConversationRole.User, Content = new
List<ContentBlock> { new ContentBlock { Text = userInput } } };
        conversation.Add(message);

        // Send the conversation to Amazon Bedrock
        var bedrockResponse = await SendConversationToBedrock(conversation);
    }
}

```

```

        // Recursively handle the model's response until the model has returned
        its final response or the recursion counter has reached 0
        await ProcessModelResponseAsync(bedrockResponse, conversation,
max_recurSIONs);

        // Repeat the loop until the user decides to exit the application
        userInput = await GetUserInputAsync();
    }

    PrintFooter();
    return conversation;
}

/// <summary>
/// Sends the conversation, the system prompt, and the tool spec to Amazon
Bedrock, and returns the response.
/// </summary>
/// <param name="conversation">The conversation history including the next
message to send.</param>
/// <returns>The response from Amazon Bedrock.</returns>
private static async Task<ConverseResponse>
SendConversationToBedrock(List<Message> conversation)
{
    Console.WriteLine("\tCalling Bedrock...");

    // Send the conversation, system prompt, and tool configuration, and return
the response
    return await _bedrockActionsWrapper.SendConverseRequestAsync(model_id,
system_prompt, conversation, _weatherTool.GetToolSpec());
}

/// <summary>
/// Processes the response received via Amazon Bedrock and performs the
necessary actions based on the stop reason.
/// </summary>
/// <param name="modelResponse">The model's response returned via Amazon
Bedrock.</param>
/// <param name="conversation">The conversation history.</param>
/// <param name="maxRecursion">The maximum number of recursive calls allowed.</
param>
private static async Task ProcessModelResponseAsync(ConverseResponse
modelResponse, List<Message> conversation, int maxRecursion)
{
    if (maxRecursion <= 0)

```

```

        {
            // Stop the process, the number of recursive calls could indicate an
infinite loop
            Console.WriteLine("\tWarning: Maximum number of recursions reached.
Please try again.");
        }

        // Append the model's response to the ongoing conversation
conversation.Add(modelResponse.Output.Message);

        if (modelResponse.StopReason == "tool_use")
        {
            // If the stop reason is "tool_use", forward everything to the tool use
handler
            await HandleToolUseAsync(modelResponse.Output, conversation,
maxRecursion - 1);
        }

        if (modelResponse.StopReason == "end_turn")
        {
            // If the stop reason is "end_turn", print the model's response text,
and finish the process
            PrintModelResponse(modelResponse.Output.Message.Content[0].Text);
            if (!_interactive)
            {
                default_prompt = "x";
            }
        }
    }

    /// <summary>
    /// Handles the tool use case by invoking the specified tool and sending the
tool's response back to Bedrock.
    /// The tool response is appended to the conversation, and the conversation is
sent back to Amazon Bedrock for further processing.
    /// </summary>
    /// <param name="modelResponse">The model's response containing the tool use
request.</param>
    /// <param name="conversation">The conversation history.</param>
    /// <param name="maxRecursion">The maximum number of recursive calls allowed.</
param>
    public static async Task HandleToolUseAsync(ConverseOutput modelResponse,
List<Message> conversation, int maxRecursion)
    {

```

```

// Initialize an empty list of tool results
var toolResults = new List<ContentBlock>();

// The model's response can consist of multiple content blocks
foreach (var contentBlock in modelResponse.Message.Content)
{
    if (!String.IsNullOrEmpty(contentBlock.Text))
    {
        // If the content block contains text, print it to the console
        PrintModelResponse(contentBlock.Text);
    }

    if (contentBlock.ToolUse != null)
    {
        // If the content block is a tool use request, forward it to the
        tool
        var toolResponse = await InvokeTool(contentBlock.ToolUse);

        // Add the tool use ID and the tool's response to the list of
        results
        toolResults.Add(new ContentBlock
        {
            ToolResult = new ToolResultBlock()
            {
                ToolUseId = toolResponse.ToolUseId,
                Content = new List<ToolResultContentBlock>()
                { new ToolResultContentBlock { Json =
toolResponse.Content } }
            }
        });
    }
}

// Embed the tool results in a new user message
var message = new Message() { Role = ConversationRole.User, Content =
toolResults };

// Append the new message to the ongoing conversation
conversation.Add(message);

// Send the conversation to Amazon Bedrock
var response = await SendConversationToBedrock(conversation);

```

```

        // Recursively handle the model's response until the model has returned its
        final response or the recursion counter has reached 0
        await ProcessModelResponseAsync(response, conversation, maxRecursion);
    }

    /// <summary>
    /// Invokes the specified tool with the given payload and returns the tool's
    response.
    /// If the requested tool does not exist, an error message is returned.
    /// </summary>
    /// <param name="payload">The payload containing the tool name and input data.</
param>
    /// <returns>The tool's response or an error message.</returns>
    public static async Task<ToolResponse> InvokeTool(ToolUseBlock payload)
    {
        var toolName = payload.Name;

        if (toolName == "Weather_Tool")
        {
            var inputData = payload.Input.AsDictionary();
            PrintToolUse(toolName, inputData);

            // Invoke the weather tool with the input data provided
            var weatherResponse = await
_weatherTool.FetchWeatherDataAsync(inputData["latitude"].ToString(),
inputData["longitude"].ToString());
            return new ToolResponse { ToolUseId = payload.ToolUseId, Content =
weatherResponse };
        }
        else
        {
            var errorMessage = $"\\tThe requested tool with name '{toolName}' does
not exist.";
            return new ToolResponse { ToolUseId = payload.ToolUseId, Content = new
{ error = true, message = errorMessage } };
        }
    }

    /// <summary>
    /// Prompts the user for input and returns the user's response.
    /// Returns null if the user enters 'x' to exit.
    /// </summary>
    /// <param name="prompt">The prompt to display to the user.</param>

```

```

    /// <returns>The user's input or null if the user chooses to exit.</returns>
    private static async Task<string?> GetUserInfoAsync(string prompt = "\tYour
weather info request:")
    {
        var userInput = default_prompt;
        if (_interactive)
        {
            Console.WriteLine(new string('*', 80));
            Console.WriteLine($"{prompt} (x to exit): \n\t");
            userInput = Console.ReadLine();
        }

        if (string.IsNullOrEmpty(userInput))
        {
            prompt = "\tPlease enter your weather info request, e.g. the name of a
city";
            return await GetUserInfoAsync(prompt);
        }

        if (userInput.ToLowerInvariant() == "x")
        {
            return null;
        }

        return userInput;
    }

    /// <summary>
    /// Logs the welcome message and usage guide for the tool use demo.
    /// </summary>
    public static void PrintHeader()
    {
        Console.WriteLine(@"
=====
Welcome to the Amazon Bedrock Tool Use demo!
=====

This assistant provides current weather information for user-specified
locations.

You can ask for weather details by providing the location name or
coordinates. Weather information
will be provided using a custom Tool and open-meteo API.

Example queries:

```

- What's the weather like in New York?
- Current weather for latitude 40.70, longitude -74.01
- Is it warmer in Rome or Barcelona today?

To exit the program, simply type 'x' and press Enter.

P.S.: You're not limited to single locations, or even to using English!
Have fun and experiment with the app!

");

}

/// <summary>

/// Logs the footer information for the tool use demo.

/// </summary>

public static void PrintFooter()

{

 Console.WriteLine(@"

=====

Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you
learned something new, or got some inspiration for your own apps today!

For more Bedrock examples in different programming languages, have a look
at:

[https://docs.aws.amazon.com/bedrock/latest/userguide/
service_code_examples.html](https://docs.aws.amazon.com/bedrock/latest/userguide/service_code_examples.html)

=====

");

}

/// <summary>

/// Logs information about the tool use.

/// </summary>

/// <param name="toolName">The name of the tool being used.</param>

/// <param name="inputData">The input data for the tool.</param>

public static void PrintToolUse(string toolName, Dictionary<string, Document>
inputData)

{

 Console.WriteLine(\$"\\n\\tInvoking tool: {toolName} with input:
{inputData["latitude"].ToString()}, {inputData["longitude"].ToString()}...\\n");

}

/// <summary>

/// Logs the model's response.

/// </summary>


```

    /// <param name="message">The model's response message.</param>
    public static void PrintModelResponse(string message)
    {
        Console.WriteLine("\tThe model's response:\n");
        Console.WriteLine(message);
        Console.WriteLine();
    }
}

```

Outil météo utilisé par la démonstration. Ce fichier définit les spécifications de l'outil et met en œuvre la logique permettant de récupérer les données météorologiques à l'aide de l'API Open-Meteo.

```

using Amazon.BedrockRuntime.Model;
using Amazon.Runtime.Documents;
using Microsoft.Extensions.Logging;

namespace ConverseToolScenario;

/// <summary>
/// Weather tool that will be invoked when requested by the Bedrock response.
/// </summary>
public class WeatherTool
{
    private readonly ILogger<WeatherTool> _logger;
    private readonly IHttpClientFactory _httpClientFactory;

    public WeatherTool(ILogger<WeatherTool> logger, IHttpClientFactory
httpClientFactory)
    {
        _logger = logger;
        _httpClientFactory = httpClientFactory;
    }

    /// <summary>
    /// Returns the JSON Schema specification for the Weather tool. The tool
specification
    /// defines the input schema and describes the tool's functionality.
    /// For more information, see https://json-schema.org/understanding-json-schema/
reference.
    /// </summary>

```

```

    /// <returns>The tool specification for the Weather tool.</returns>
    public ToolSpecification GetToolSpec()
    {
        ToolSpecification toolSpecification = new ToolSpecification();

        toolSpecification.Name = "Weather_Tool";
        toolSpecification.Description = "Get the current weather for a given
location, based on its WGS84 coordinates.";

        Document toolSpecDocument = Document.FromObject(
            new
            {
                type = "object",
                properties = new
                {
                    latitude = new
                    {
                        type = "string",
                        description = "Geographical WGS84 latitude of the location."
                    },
                    longitude = new
                    {
                        type = "string",
                        description = "Geographical WGS84 longitude of the
location."
                    }
                },
                required = new[] { "latitude", "longitude" }
            });

        toolSpecification.InputSchema = new ToolInputSchema() { Json =
toolSpecDocument };
        return toolSpecification;
    }

    /// <summary>
    /// Fetches weather data for the given latitude and longitude using the Open-
    Meteo API.
    /// Returns the weather data or an error message if the request fails.
    /// </summary>
    /// <param name="latitude">The latitude of the location.</param>
    /// <param name="longitude">The longitude of the location.</param>
    /// <returns>The weather data or an error message.</returns>

```

```

    public async Task<Document> FetchWeatherDataAsync(string latitude, string
longitude)
    {
        string endpoint = "https://api.open-meteo.com/v1/forecast";

        try
        {
            var httpClient = _httpClientFactory.CreateClient();
            var response = await httpClient.GetAsync($"{endpoint}?
latitude={latitude}&longitude={longitude}&current_weather=True");
            response.EnsureSuccessStatusCode();
            var weatherData = await response.Content.ReadAsStringAsync();

            Document weatherDocument = Document.FromObject(
                new { weather_data = weatherData });

            return weatherDocument;
        }
        catch (HttpRequestException e)
        {
            _logger.LogError(e, "Error fetching weather data: {Message}",
e.Message);
            throw;
        }
        catch (Exception e)
        {
            _logger.LogError(e, "Unexpected error fetching weather data: {Message}",
e.Message);
            throw;
        }
    }
}

```

Action de l'API Converse avec une configuration d'outil.

```

/// <summary>
/// Wrapper class for interacting with the Amazon Bedrock Converse API.
/// </summary>
public class BedrockActionsWrapper
{
    private readonly IAmazonBedrockRuntime _bedrockClient;

```

```

private readonly ILogger<BedrockActionsWrapper> _logger;

/// <summary>
/// Initializes a new instance of the <see cref="BedrockActionsWrapper"/> class.
/// </summary>
/// <param name="bedrockClient">The Bedrock Converse API client.</param>
/// <param name="logger">The logger instance.</param>
public BedrockActionsWrapper(IAmazonBedrockRuntime bedrockClient,
ILogger<BedrockActionsWrapper> logger)
{
    _bedrockClient = bedrockClient;
    _logger = logger;
}

/// <summary>
/// Sends a Converse request to the Amazon Bedrock Converse API.
/// </summary>
/// <param name="modelId">The Bedrock Model Id.</param>
/// <param name="systemPrompt">A system prompt instruction.</param>
/// <param name="conversation">The array of messages in the conversation.</
param>
/// <param name="toolSpec">The specification for a tool.</param>
/// <returns>The response of the model.</returns>
public async Task<ConverseResponse> SendConverseRequestAsync(string modelId,
string systemPrompt, List<Message> conversation, ToolSpecification toolSpec)
{
    try
    {
        var request = new ConverseRequest()
        {
            ModelId = modelId,
            System = new List<SystemContentBlock>()
            {
                new SystemContentBlock()
                {
                    Text = systemPrompt
                }
            },
            Messages = conversation,
            ToolConfig = new ToolConfiguration()
            {
                Tools = new List<Tool>()
                {
                    new Tool()

```

```
        {
            ToolSpec = toolSpec
        }
    }
};

var response = await _bedrockClient.ConverseAsync(request);

return response;
}
catch (ModelNotReadyException ex)
{
    _logger.LogError(ex, "Model not ready, please wait and try again.");
    throw;
}
catch (AmazonBedrockRuntimeException ex)
{
    _logger.LogError(ex, "Error occurred while sending Converse request.");
    throw;
}
}
```

- Pour plus de détails sur l'API, consultez [Converse](#) dans la Référence des API du kit AWS SDK pour .NET .

Amazon Nova Canvas

InvokeModel

L'exemple de code suivant montre comment invoquer Amazon Nova Canvas sur Amazon Bedrock pour générer une image.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une image avec Amazon Nova Canvas.

```
// Use the native inference API to create an image with Amazon Nova Canvas.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID.
var modelId = "amazon.nova-canvas-v1:0";

// Define the image generation prompt for the model.
var prompt = "A stylized picture of a cute old steampunk robot.";

// Create a random seed between 0 and 858,993,459
int seed = new Random().Next(0, 858993460);

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    taskType = "TEXT_IMAGE",
    textToImageParams = new
    {
        text = prompt
    },
    imageGenerationConfig = new
    {
        seed,
        quality = "standard",
        width = 512,
        height = 512,
        numberOfImages = 1
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
```

```
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract the image data.
    var base64Image = modelResponse["images"]?[0].ToString() ?? "";

    // Save the image in a local folder
    string savedPath = AmazonNovaCanvas.InvokeModel.SaveBase64Image(base64Image);
    Console.WriteLine($"Image saved to: {savedPath}");
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModel](#) à la section Référence des AWS SDK pour .NET API.

Anthropic Claude

InvokeModelWithResponseStream

L'exemple de code suivant montre comment envoyer un message texte aux modèles Anthropic Claude à l'aide de l'API Invoke Model et imprimer le flux de réponses.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

```
// Use the native inference API to send a text message to Anthropic Claude
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});
```



```
// Create a request with the model ID, the user message, and an inference
// configuration.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
    client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
        PayloadPart).Bytes);
        var text = chunk["delta"]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModelWithResponseStream](#) à la section Référence des AWS SDK pour .NET API.

Cohere Command

InvokeModel: Command R et R+

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command R et R+ à l'aide de l'API Invoke Model.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte.

```
// Use the native inference API to send a text message to Cohere Command R.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
```

```
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModel](#) à la section Référence des AWS SDK pour .NET API.

InvokeModelWithResponseStream: Command R et R+

L'exemple de code suivant montre comment envoyer un message texte à Cohere Command, à l'aide de l'API Invoke Model avec un flux de réponse.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

```
// Use the native inference API to send a text message to Cohere Command R
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
        client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
```

```
foreach (var item in streamingResponse.Body)
{
    var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
    var text = chunk["text"] ?? "";
    Console.Write(text);
}
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModel](#) à la section Référence des AWS SDK pour .NET API.

Meta Llama

InvokeModel

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama, à l'aide de l'API Invoke Model.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte.

```
// Use the native inference API to send a text message to Meta Llama 3.

using System;
using System.IO;
using System.Text.Json;
```

```
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USSouth2);

// Set the model ID, e.g., Llama 3 70b Instruct.
var modelId = "meta.llama3-70b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
```

```
var modelResponse = await JsonNode.ParseAsync(response.Body);

// Extract and print the response text.
var responseText = modelResponse["generation"] ?? "";
Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModel](#) à la section Référence des AWS SDK pour .NET API.

InvokeModelWithResponseStream

L'exemple de code suivant montre comment envoyer un message texte à Meta Llama à l'aide de l'API Invoke Model et imprimer le flux de réponses.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

```
// Use the native inference API to send a text message to Meta Llama 3
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
```

```

using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USWest2);

// Set the model ID, e.g., Llama 3 70b Instruct.
var modelId = "meta.llama3-70b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)

```



```
{
    var chunk = JsonSerializer.Deserialize<JsonObject>((item as
PayloadPart).Bytes);
    var text = chunk["generation"] ?? "";
    Console.Write(text);
}
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModelWithResponseStream](#) à la section Référence des AWS SDK pour .NET API.

Mistral AI

InvokeModel

L'exemple de code suivant montre comment envoyer un message texte aux modèles Mistral, à l'aide de l'API Invoke Model.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte.

```
// Use the native inference API to send a text message to Mistral.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
```

```
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["outputs"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
```

```
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModel](#) à la section Référence des AWS SDK pour .NET API.

InvokeModelWithResponseStream

L'exemple de code suivant montre comment envoyer un message texte aux modèles Mistral AI à l'aide de l'API Invoke Model et imprimer le flux de réponses.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez l'API Invoke Model pour envoyer un message texte et traiter le flux de réponses en temps réel.

```
// Use the native inference API to send a text message to Mistral
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);
```

```
// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
        client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
        PayloadPart).Bytes);
        var text = chunk["outputs"]?[0]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [InvokeModelWithResponseStream](#) à la section Référence des AWS SDK pour .NET API.

CloudWatch Exemples de journaux utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with CloudWatch Logs.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

AssociateKmsKey

L'exemple de code suivant montre comment utiliser `AssociateKmsKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;
```

```
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to associate an AWS Key Management Service (AWS KMS) key with
/// an Amazon CloudWatch Logs log group.
/// </summary>
public class AssociateKmsKey
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
        string groupName = "cloudwatchlogs-example-loggroup";

        var request = new AssociateKmsKeyRequest
        {
            KmsKeyId = kmsKeyId,
            LogGroupName = groupName,
        };

        var response = await client.AssociateKmsKeyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully associated KMS key ID: {kmsKeyId}
with log group: {groupName}.");
        }
        else
        {
            Console.WriteLine("Could not make the association between:
{kmsKeyId} and {groupName}.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [AssociateKmsKey](#) à la section Référence des AWS SDK pour .NET API.

CancelExportTask

L'exemple de code suivant montre comment utiliser `CancelExportTask`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CancelExportTask](#) à la section Référence des AWS SDK pour .NET API.

CreateExportTask

L'exemple de code suivant montre comment utiliser `CreateExportTask`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {

```



```
// This client object will be associated with the same AWS Region
// as the default user on this system. If you need to use a
// different AWS Region, pass it as a parameter to the client
// constructor.
var client = new AmazonCloudWatchLogsClient();
string taskName = "export-task-example";
string logGroupName = "cloudwatchlogs-example-loggroup";
string destination = "amzn-s3-demo-bucket";
var fromTime = 1437584472382;
var toTime = 1437584472833;

var request = new CreateExportTaskRequest
{
    From = fromTime,
    To = toTime,
    TaskName = taskName,
    LogGroupName = logGroupName,
    Destination = destination,
};

var response = await client.CreateExportTaskAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"The task, {taskName} with ID: " +
        $"{response.TaskId} has been created
successfully.");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateExportTask](#) à la section Référence des AWS SDK pour .NET API.

CreateLogGroup

L'exemple de code suivant montre comment utiliser `CreateLogGroup`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to create an Amazon CloudWatch Logs log group.
/// </summary>
public class CreateLogGroup
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new CreateLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.CreateLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
        }
        else
        {

```

```
        Console.WriteLine("Could not create log group.");  
    }  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateLogGroup](#) à la section Référence des AWS SDK pour .NET API.

CreateLogStream

L'exemple de code suivant montre comment utiliser `CreateLogStream`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;  
using Amazon.CloudWatchLogs.Model;  
  
/// <summary>  
/// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch  
/// log group.  
/// </summary>  
public class CreateLogStream  
{  
    public static async Task Main()  
    {  
        // This client object will be associated with the same AWS Region  
        // as the default user on this system. If you need to use a  
        // different AWS Region, pass it as a parameter to the client  
        // constructor.  
        var client = new AmazonCloudWatchLogsClient();  
        string logGroupName = "cloudwatchlogs-example-loggroup";
```

```
string logStreamName = "cloudwatchlogs-example-logstream";

var request = new CreateLogStreamRequest
{
    LogGroupName = logGroupName,
    LogStreamName = logStreamName,
};

var response = await client.CreateLogStreamAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"{logStreamName} successfully created for {logGroupName}.");
}
else
{
    Console.WriteLine("Could not create stream.");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateLogStream](#) à la section Référence des AWS SDK pour .NET API.

DeleteLogGroup

L'exemple de code suivant montre comment utiliser `DeleteLogGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
```

```
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Uses the Amazon CloudWatch Logs Service to delete an existing
/// CloudWatch Logs log group.
/// </summary>
public class DeleteLogGroup
{
    public static async Task Main()
    {
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new DeleteLogGroupRequest
        {
            LogGroupName = logGroupName,
        };

        var response = await client.DeleteLogGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteLogGroup](#) à la section Référence des AWS SDK pour .NET API.

DescribeExportTasks

L'exemple de code suivant montre comment utiliser `DescribeExportTasks`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

/// <summary>
/// Shows how to retrieve a list of information about Amazon CloudWatch
/// Logs export tasks.
/// </summary>
public class DescribeExportTasks
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        var request = new DescribeExportTasksRequest
        {
            Limit = 5,
        };

        var response = new DescribeExportTasksResponse();

        do
        {
            response = await client.DescribeExportTasksAsync(request);
            response.ExportTasks.ForEach(t =>
            {
                Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has status:
{t.Status}");
            });
        }
```

```
        }  
        while (response.NextToken is not null);  
    }  
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeExportTasks](#) à la section Référence des AWS SDK pour .NET API.

DescribeLogGroups

L'exemple de code suivant montre comment utiliser `DescribeLogGroups`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.CloudWatchLogs;  
using Amazon.CloudWatchLogs.Model;  
  
/// <summary>  
/// Retrieves information about existing Amazon CloudWatch Logs log groups  
/// and displays the information on the console.  
/// </summary>  
public class DescribeLogGroups  
{  
    public static async Task Main()  
    {  
        // Creates a CloudWatch Logs client using the default  
        // user. If you need to work with resources in another  
        // AWS Region than the one defined for the default user,  
        // pass the AWS Region as a parameter to the client constructor.  
        var client = new AmazonCloudWatchLogsClient();
```

```
bool done = false;
string newToken = null;

var request = new DescribeLogGroupsRequest
{
    Limit = 5,
};

DescribeLogGroupsResponse response;

do
{
    if (newToken is not null)
    {
        request.NextToken = newToken;
    }

    response = await client.DescribeLogGroupsAsync(request);

    response.LogGroups.ForEach(lg =>
    {
        Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
        Console.WriteLine($"Created on: {lg.CreationTime.Date.Date}");
        Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
    });

    if (response.NextToken is null)
    {
        done = true;
    }
    else
    {
        newToken = response.NextToken;
    }
}
while (!done);
}
```


- Pour plus de détails sur l'API, reportez-vous [DescribeLogGroups](#) à la section Référence des AWS SDK pour .NET API.

StartLiveTail

L'exemple de code suivant montre comment utiliser `StartLiveTail`.

SDK pour .NET

Joignez les fichiers requis.

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Démarrez une session Live Tail.

```
var client = new AmazonCloudWatchLogsClient();
var request = new StartLiveTailRequest
{
    LogGroupIdentifiers = logGroupIdentifiers,
    LogStreamNames = logStreamNames,
    LogEventFilterPattern = filterPattern,
};

var response = await client.StartLiveTailAsync(request);

// Catch if request fails
if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
{
    Console.WriteLine("Failed to start live tail session");
    return;
}
```

Vous pouvez gérer les événements de la session Live Tail de deux manières :

```
/* Method 1
 * 1). Asynchronously loop through the event stream
 * 2). Set a timer to dispose the stream and stop the Live Tail session
at the end.
```

```

    */
    var eventStream = response.ResponseStream;
    var task = Task.Run(() =>
    {
        foreach (var item in eventStream)
        {
            if (item is LiveTailSessionUpdate liveTailSessionUpdate)
            {
                foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
                {
                    Console.WriteLine("Message : {0}",
sessionResult.Message);
                }
            }
            if (item is LiveTailSessionStart)
            {
                Console.WriteLine("Live Tail session started");
            }
            // On-stream exceptions are processed here
            if (item is CloudWatchLogsEventStreamException)
            {
                Console.WriteLine($"ERROR: {item}");
            }
        }
    });
    // Close the stream to stop the session after a timeout
    if (!task.Wait(TimeSpan.FromSeconds(10))){
        eventStream.Dispose();
        Console.WriteLine("End of line");
    }

```

```

    /* Method 2
    * 1). Add event handlers to each event variable
    * 2). Start processing the stream and wait for a timeout using
AutoResetEvent
    */
    AutoResetEvent endEvent = new AutoResetEvent(false);
    var eventStream = response.ResponseStream;
    using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
    {

```

```
eventStream.SessionStartReceived += (sender, e) =>
{
    Console.WriteLine("LiveTail session started");
};
eventStream.SessionUpdateReceived += (sender, e) =>
{
    foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
        Console.WriteLine("Message: {0}", logEvent.Message);
    }
};
// On-stream exceptions are captured here
eventStream.ExceptionReceived += (sender, e) =>
{
    Console.WriteLine($"ERROR: {e.EventStreamException.Message}");
};

eventStream.StartProcessing();
// Stream events for this amount of time.
endEvent.WaitOne(TimeSpan.FromSeconds(10));
Console.WriteLine("End of line");
}
```

- Pour plus de détails sur l'API, reportez-vous [StartLiveTail](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation du fournisseur d'identité Amazon Cognito SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du fournisseur AWS SDK pour .NET d'identité Amazon Cognito.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

AdminGetUser

L'exemple de code suivant montre comment utiliser `AdminGetUser`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the specified user from an Amazon Cognito user pool with administrator
access.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
/// <returns>Async task.</returns>
public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
{
    AdminGetUserRequest userRequest = new AdminGetUserRequest
    {
        Username = userName,
        UserPoolId = poolId,
    };

    var response = await _cognitoService.AdminGetUserAsync(userRequest);

    Console.WriteLine($"User status {response.UserStatus}");
}
```

```
        return response.UserStatus;
    }
```

- Pour plus de détails sur l'API, reportez-vous [AdminGetUser](#) à la section Référence des AWS SDK pour .NET API.

AdminInitiateAuth

L'exemple de code suivant montre comment utiliser `AdminInitiateAuth`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Initiate an admin auth request.
/// </summary>
/// <param name="clientId">The client ID to use.</param>
/// <param name="userPoolId">The ID of the user pool.</param>
/// <param name="userName">The username to authenticate.</param>
/// <param name="password">The user's password.</param>
/// <returns>The session to use in challenge-response.</returns>
public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var request = new AdminInitiateAuthRequest
    {
        ClientId = clientId,
        UserPoolId = userPoolId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
```

```
};

var response = await _cognitoService.AdminInitiateAuthAsync(request);
return response.Session;
}
```

- Pour plus de détails sur l'API, reportez-vous [AdminInitiateAuth](#) à la section Référence des AWS SDK pour .NET API.

AdminRespondToAuthChallenge

L'exemple de code suivant montre comment utiliser `AdminRespondToAuthChallenge`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
```

```

challengeResponses.Add("USERNAME", userName);
challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
{
    ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ClientId = clientId,
    ChallengeResponses = challengeResponses,
    Session = session,
    UserPoolId = userPoolId,
};

var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
return response.AuthenticationResult;
}

```

- Pour plus de détails sur l'API, reportez-vous [AdminRespondToAuthChallenge](#) à la section Référence des AWS SDK pour .NET API.

AssociateSoftwareToken

L'exemple de code suivant montre comment utiliser `AssociateSoftwareToken`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get an MFA token to authenticate the user with the authenticator.
/// </summary>
/// <param name="session">The session name.</param>
/// <returns>The session name.</returns>

```

```
public async Task<string> AssociateSoftwareTokenAsync(string session)
{
    var softwareTokenRequest = new AssociateSoftwareTokenRequest
    {
        Session = session,
    };

    var tokenResponse = await
        _cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
    var secretCode = tokenResponse.SecretCode;

    Console.WriteLine($"Use the following secret code to set up the
    authenticator: {secretCode}");

    return tokenResponse.Session;
}
```

- Pour plus de détails sur l'API, reportez-vous [AssociateSoftwareToken](#) à la section Référence des AWS SDK pour .NET API.

ConfirmDevice

L'exemple de code suivant montre comment utiliser `ConfirmDevice`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Initiates and confirms tracking of the device.
/// </summary>
/// <param name="accessToken">The user's access token.</param>
/// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
/// <param name="deviceName">The device name.</param>
/// <returns></returns>
```



```
public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
{
    var request = new ConfirmDeviceRequest
    {
        AccessToken = accessToken,
        DeviceKey = deviceKey,
        DeviceName = deviceName
    };

    var response = await _cognitoService.ConfirmDeviceAsync(request);
    return response.UserConfirmationNecessary;
}
```

- Pour plus de détails sur l'API, reportez-vous [ConfirmDevice](#) à la section Référence des AWS SDK pour .NET API.

ConfirmSignUp

L'exemple de code suivant montre comment utiliser `ConfirmSignUp`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Confirm that the user has signed up.
/// </summary>
/// <param name="clientId">The Id of this application.</param>
/// <param name="code">The confirmation code sent to the user.</param>
/// <param name="userName">The username.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
{
```

```
var signUpRequest = new ConfirmSignUpRequest
{
    ClientId = clientId,
    ConfirmationCode = code,
    Username = userName,
};

var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
if (response.HttpStatusCode == HttpStatusCode.OK)
{
    Console.WriteLine($"{userName} was confirmed");
    return true;
}
return false;
}
```

- Pour plus de détails sur l'API, reportez-vous [ConfirmSignUp](#) à la section Référence des AWS SDK pour .NET API.

InitiateAuth

L'exemple de code suivant montre comment utiliser `InitiateAuth`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Initiate authorization.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The name of the user who is authenticating.</param>
/// <param name="password">The password for the user who is authenticating.</param>
/// <returns>The response from the initiate auth request.</returns>
```

```
public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var authRequest = new InitiateAuthRequest
    {
        ClientId = clientId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.InitiateAuthAsync(authRequest);
    Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

    return response;
}
```

- Pour plus de détails sur l'API, reportez-vous [InitiateAuth](#) à la section Référence des AWS SDK pour .NET API.

ListUsers

L'exemple de code suivant montre comment utiliser `ListUsers`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of users for the Amazon Cognito user pool.
/// </summary>
/// <param name="userPoolId">The user pool ID.</param>
```

```
/// <returns>A list of users.</returns>
public async Task<List<UserType>> ListUsersAsync(string userPoolId)
{
    var request = new ListUsersRequest
    {
        UserPoolId = userPoolId
    };

    var users = new List<UserType>();

    var usersPaginator = _cognitoService.Paginators.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListUsers](#) à la section Référence des AWS SDK pour .NET API.

ResendConfirmationCode

L'exemple de code suivant montre comment utiliser `ResendConfirmationCode`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Send a new confirmation code to a user.
/// </summary>
/// <param name="clientId">The Id of the client application.</param>
```

```
/// <param name="userName">The username of user who will receive the code.</param>
/// <returns>The delivery details.</returns>
public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
clientId, string userName)
{
    var codeRequest = new ResendConfirmationCodeRequest
    {
        ClientId = clientId,
        Username = userName,
    };

    var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);

    Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

    return response.CodeDeliveryDetails;
}
```

- Pour plus de détails sur l'API, reportez-vous [ResendConfirmationCode](#) à la section Référence des AWS SDK pour .NET API.

SignUp

L'exemple de code suivant montre comment utiliser `SignUp`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Sign up a new user.
/// </summary>
```

```
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The username to use.</param>
/// <param name="password">The user's password.</param>
/// <param name="email">The email address of the user.</param>
/// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Pour plus de détails sur l'API, reportez-vous [SignUp](#) à la section Référence des AWS SDK pour .NET API.

VerifySoftwareToken

L'exemple de code suivant montre comment utiliser `VerifySoftwareToken`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Verify the TOTP and register for MFA.
/// </summary>
/// <param name="session">The name of the session.</param>
/// <param name="code">The MFA code.</param>
/// <returns>The status of the software token.</returns>
public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
{
    var tokenRequest = new VerifySoftwareTokenRequest
    {
        UserCode = code,
        Session = session,
    };

    var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

    return verifyResponse.Status;
}
```

- Pour plus de détails sur l'API, reportez-vous [VerifySoftwareToken](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Inscription d'un utilisateur auprès d'un groupe d'utilisateurs nécessitant l'authentification MFA

L'exemple de code suivant illustre comment :

- Inscrivez et confirmez un utilisateur avec un nom d'utilisateur, un mot de passe et une adresse e-mail.
- Configurez l'authentification multifactorielle en associant une application MFA à l'utilisateur.
- Connectez-vous à l'aide d'un mot de passe et d'un code MFA.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Cognito.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCognitoIdentityProvider>()
                    .AddTransient<CognitoWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CognitoBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
```



```
        true) // Optionally load local settings.
        .Build();

var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

Console.WriteLine(new string('-', 80));
UiMethods.DisplayOverview();
Console.WriteLine(new string('-', 80));

// clientId - The app client Id value that you get from the AWS CDK script.
var clientId = configuration["ClientId"]; // **** REPLACE WITH CLIENT ID
VALUE FROM CDK SCRIPT";

// poolId - The pool Id that you get from the AWS CDK script.
var poolId = configuration["PoolId"]!; // **** REPLACE WITH POOL ID VALUE
FROM CDK SCRIPT";
var userName = configuration["UserName"];
var password = configuration["Password"];
var email = configuration["Email"];

// If the username wasn't set in the configuration file,
// get it from the user now.
if (userName is null)
{
    do
    {
        Console.Write("Username: ");
        userName = Console.ReadLine();
    }
    while (string.IsNullOrEmpty(userName));
}
Console.WriteLine($"\\nUsername: {userName}");

// If the password wasn't set in the configuration file,
// get it from the user now.
if (password is null)
{
    do
    {
        Console.Write("Password: ");
        password = Console.ReadLine();
    }
    while (string.IsNullOrEmpty(password));
}
```

```
// If the email address wasn't set in the configuration file,
// get it from the user now.
if (email is null)
{
    do
    {
        Console.Write("Email: ");
        email = Console.ReadLine();
    } while (string.IsNullOrEmpty(email));
}

// Now sign up the user.
Console.WriteLine($"Signing up {userName} with email address: {email}");
await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

// Add the user to the user pool.
Console.WriteLine($"Adding {userName} to the user pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

UiMethods.DisplayTitle("Get confirmation code");
Console.WriteLine($"Conformation code sent to {userName}.");
Console.Write("Would you like to send a new code? (Y/N) ");
var answer = Console.ReadLine();

if (answer!.ToLower() == "y")
{
    await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
    Console.WriteLine("Sending a new confirmation code");
}

Console.Write("Enter confirmation code (from Email): ");
var code = Console.ReadLine();

await cognitoWrapper.ConfirmSignupAsync(clientId, code, userName);

UiMethods.DisplayTitle("Checking status");
Console.WriteLine($"Rechecking the status of {userName} in the user pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

Console.WriteLine($"Setting up authenticator for {userName} in the user
pool");
var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
userName, password);
```

```

        var setupSession = await
cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
        Console.WriteLine("Enter the 6-digit code displayed in Google Authenticator: ");
        var setupCode = Console.ReadLine();

        var setupResult = await
cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
        Console.WriteLine($"Setup status: {setupResult}");

        Console.WriteLine($"Now logging in {userName} in the user pool");
        var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
poolId, userName, password);

        Console.WriteLine("Enter a new 6-digit code displayed in Google Authenticator:
");
        var authCode = Console.ReadLine();

        var authResult = await
cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
authSession, poolId);
        Console.WriteLine($"Authenticated and received access token:
{authResult.AccessToken}");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cognito scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
}

using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
    /// Constructor for the wrapper class containing Amazon Cognito actions.

```

```

    /// </summary>
    /// <param name="cognitoService">The Amazon Cognito client object.</param>
    public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)
    {
        _cognitoService = cognitoService;
    }

    /// <summary>
    /// List the Amazon Cognito user pools for an account.
    /// </summary>
    /// <returns>A list of UserPoolDescriptionType objects.</returns>
    public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
    {
        var userPools = new List<UserPoolDescriptionType>();

        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }

    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
        {

```

```

        users.AddRange(response.Users);
    }

    return users;
}

/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
    challengeResponses.Add("USERNAME", userName);
    challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
    {
        ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
        ClientId = clientId,
        ChallengeResponses = challengeResponses,
        Session = session,
        UserPoolId = userPoolId,
    };

    var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
    Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
    return response.AuthenticationResult;
}

```

```

    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
    }

    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }

```

```

    }

    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }

    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);
    }

```

```

    var authRequest = new InitiateAuthRequest

    {
        ClientId = clientId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.InitiateAuthAsync(authRequest);
    Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

    return response;
}

/// <summary>
/// Confirm that the user has signed up.
/// </summary>
/// <param name="clientId">The Id of this application.</param>
/// <param name="code">The confirmation code sent to the user.</param>
/// <param name="userName">The username.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
{
    var signUpRequest = new ConfirmSignUpRequest
    {
        ClientId = clientId,
        ConfirmationCode = code,
        Username = userName,
    };

    var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
    if (response.HttpStatusCode == HttpStatusCode.OK)
    {
        Console.WriteLine($"{userName} was confirmed");
        return true;
    }
    return false;
}

/// <summary>
/// Initiates and confirms tracking of the device.

```



```

    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }

    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);

        Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }

```

```

    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with administrator
    access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
    poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }

    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
    returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
    password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

```

```
        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Exemples d'Amazon Comprehend utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Comprehend.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

DetectDominantLanguage

L'exemple de code suivant montre comment utiliser DetectDominantLanguage.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;
```

```

    /// <summary>
    /// This example calls the Amazon Comprehend service to determine the
    /// dominant language.
    /// </summary>
    public static class DetectDominantLanguage
    {
        /// <summary>
        /// Calls Amazon Comprehend to determine the dominant language used in
        /// the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle.";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            Console.WriteLine("Calling DetectDominantLanguage\n");
            var detectDominantLanguageRequest = new DetectDominantLanguageRequest()
            {
                Text = text,
            };

            var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
            foreach (var dl in detectDominantLanguageResponse.Languages)
            {
                Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
            }

            Console.WriteLine("Done");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DetectDominantLanguage](#) à la section Référence des AWS SDK pour .NET API.

DetectEntities

L'exemple de code suivant montre comment utiliser `DetectEntities`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use the AmazonComprehend service detect any
/// entities in submitted text.
/// </summary>
public static class DetectEntities
{
    /// <summary>
    /// The main method calls the DetectEntitiesAsync method to find any
    /// entities in the sample code.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new AmazonComprehendClient();

        Console.WriteLine("Calling DetectEntities\n");
        var detectEntitiesRequest = new DetectEntitiesRequest()
        {
            Text = text,
            LanguageCode = "en",
        };
        var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

        foreach (var e in detectEntitiesResponse.Entities)
        {
            Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
        }
    }
}
```

```
    }  
  
    Console.WriteLine("Done");  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectEntities](#) à la section Référence des AWS SDK pour .NET API.

DetectKeyPhrases

L'exemple de code suivant montre comment utiliser `DetectKeyPhrases`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Comprehend;  
using Amazon.Comprehend.Model;  
  
/// <summary>  
/// This example shows how to use the Amazon Comprehend service to  
/// search text for key phrases.  
/// </summary>  
public static class DetectKeyPhrase  
{  
    /// <summary>  
    /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync  
    /// to detect any key phrases in the sample text.  
    /// </summary>  
    public static async Task Main()  
    {  
        string text = "It is raining today in Seattle";
```

```
var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USSouth1);

// Call DetectKeyPhrases API
Console.WriteLine("Calling DetectKeyPhrases");
var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
{
    Text = text,
    LanguageCode = "en",
};
var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
{
    Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score}, BeginOffset:
{kp.BeginOffset}, EndOffset: {kp.EndOffset}");
}

Console.WriteLine("Done");
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectKeyPhrases](#) à la section Référence des AWS SDK pour .NET API.

DetectPiiEntities

L'exemple de code suivant montre comment utiliser `DetectPiiEntities`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
```



```
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally identifiable information within the supplied text.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();
        var text = @"Hello Paul Santos. The latest statement for your
                    credit card account 1111-0000-1111-0000 was
                    mailed to 123 Any Street, Seattle, WA 98109.";

        var request = new DetectPiiEntitiesRequest
        {
            Text = text,
            LanguageCode = "EN",
        };

        var response = await comprehendClient.DetectPiiEntitiesAsync(request);

        if (response.Entities.Count > 0)
        {
            foreach (var entity in response.Entities)
            {
                var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                Console.WriteLine($"{entity.Type}: {entityValue}");
            }
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectPiiEntities](#) à la section Référence des AWS SDK pour .NET API.

DetectSentiment

L'exemple de code suivant montre comment utiliser `DetectSentiment`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to detect the overall sentiment of the supplied
/// text using the Amazon Comprehend service.
/// </summary>
public static class DetectSentiment
{
    /// <summary>
    /// This method calls the DetectSentimentAsync method to analyze the
    /// supplied text and determine the overall sentiment.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

        // Call DetectKeyPhrases API
        Console.WriteLine("Calling DetectSentiment");
        var detectSentimentRequest = new DetectSentimentRequest()
        {
            Text = text,
```

```

        LanguageCode = "en",
    };
    var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
    Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
    Console.WriteLine("Done");
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DetectSentiment](#) à la section Référence des AWS SDK pour .NET API.

DetectSyntax

L'exemple de code suivant montre comment utiliser DetectSyntax.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use Amazon Comprehend to detect syntax
/// elements by calling the DetectSyntaxAsync method.
/// </summary>
public class DetectingSyntax
{
    /// <summary>
    /// This method calls DetectSynaxAsync to identify the syntax elements
    /// in the sample text.
    /// </summary>

```

```
public static async Task Main()
{
    string text = "It is raining today in Seattle";

    var comprehendClient = new AmazonComprehendClient();

    // Call DetectSyntax API
    Console.WriteLine("Calling DetectSyntaxAsync\n");
    var detectSyntaxRequest = new DetectSyntaxRequest()
    {
        Text = text,
        LanguageCode = "en",
    };
    DetectSyntaxResponse detectSyntaxResponse = await
comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
    foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
    {
        Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
{s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
    }

    Console.WriteLine("Done");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectSyntax](#) à la section Référence des AWS SDK pour .NET API.

StartTopicsDetectionJob

L'exemple de code suivant montre comment utiliser `StartTopicsDetectionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example scans the documents in an Amazon Simple Storage Service
/// (Amazon S3) bucket and analyzes it for topics. The results are stored
/// in another bucket and then the resulting job properties are displayed
/// on the screen. This example was created using the AWS SDK for .NET
/// version 3.7 and .NET Core version 5.0.
/// </summary>
public static class TopicModeling
{
    /// <summary>
    /// This methos calls a topic detection job by calling the Amazon
    /// Comprehend StartTopicsDetectionJobRequest.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();

        string inputS3Uri = "s3://input bucket/input path";
        InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
        string outputS3Uri = "s3://output bucket/output path";
        string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";

        int numberOfTopics = 10;

        var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
        {
            InputDataConfig = new InputDataConfig()
            {
                S3Uri = inputS3Uri,
                InputFormat = inputDocFormat,
            },
            OutputDataConfig = new OutputDataConfig()
            {
                S3Uri = outputS3Uri,
            },
            DataAccessRoleArn = dataAccessRoleArn,
            NumberOfTopics = numberOfTopics,
        }
    }
}

```

```

    };

    var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

    var jobId = startTopicsDetectionJobResponse.JobId;
    Console.WriteLine("JobId: " + jobId);

    var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
    {
        JobId = jobId,
    };

    var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);

PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

    var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
    foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
    {
        PrintJobProperties(props);
    }
}

/// <summary>
/// This method is a helper method that displays the job properties
/// from the call to StartTopicsDetectionJobRequest.
/// </summary>
/// <param name="props">A list of properties from the call to
/// StartTopicsDetectionJobRequest.</param>
private static void PrintJobProperties(TopicsDetectionJobProperties props)
{
    Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
    Console.WriteLine($"NumberOfTopics: {props.NumberOfTopics}\nInputS3Uri:
{props.InputDataConfig.S3Uri}");
    Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
}

```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [StartTopicsDetectionJob](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemples d'Amazon DocumentDB utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon DocumentDB.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Exemples sans serveur](#)

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon DocumentDB

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux de modifications DocumentDB. La fonction récupère les données utiles DocumentDB et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Amazon DocumentDB avec Lambda en utilisant .NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
//Assembly attribute to enable the Lambda function's JSON input to be converted into
//a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
```



```
namespace LambdaDocDb;

public class Function
{
    /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

    private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
    ILambdaContext context)
    {
        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument, new
        JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }

    public class Event
    {
        [JsonPropertyName("eventSourceArn")]
    }
}
```

```
        public string EventSourceArn { get; set; }

        [JsonPropertyName("events")]
        public List<DocumentDBEventRecord> Events { get; set; }

        [JsonPropertyName("eventSource")]
        public string EventSource { get; set; }
    }

    public class DocumentDBEventRecord
    {
        [JsonPropertyName("event")]
        public EventData Event { get; set; }
    }

    public class EventData
    {
        [JsonPropertyName("_id")]
        public IdData Id { get; set; }

        [JsonPropertyName("clusterTime")]
        public ClusterTime ClusterTime { get; set; }

        [JsonPropertyName("documentKey")]
        public DocumentKey DocumentKey { get; set; }

        [JsonPropertyName("fullDocument")]
        public Dictionary<string, object> FullDocument { get; set; }

        [JsonPropertyName("ns")]
        public Namespace Namespace { get; set; }

        [JsonPropertyName("operationType")]
        public string OperationType { get; set; }
    }

    public class IdData
    {
        [JsonPropertyName("_data")]
        public string Data { get; set; }
    }

    public class ClusterTime
    {
```

```
        [JsonPropertyName("$timestamp")]
        public Timestamp Timestamp { get; set; }
    }

    public class Timestamp
    {
        [JsonPropertyName("t")]
        public long T { get; set; }

        [JsonPropertyName("i")]
        public int I { get; set; }
    }

    public class DocumentKey
    {
        [JsonPropertyName("_id")]
        public Id Id { get; set; }
    }

    public class Id
    {
        [JsonPropertyName("$oid")]
        public string Oid { get; set; }
    }

    public class Namespace
    {
        [JsonPropertyName("db")]
        public string Db { get; set; }

        [JsonPropertyName("coll")]
        public string Coll { get; set; }
    }
}
```

Exemples DynamoDB utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide de DynamoDB.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

AWS les contributions communautaires sont des exemples qui ont été créés et sont maintenus par plusieurs équipes AWS. Pour faire un commentaire, utilisez le mécanisme fourni dans les référentiels liés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)
- [AWS contributions communautaires](#)

Actions

BatchExecuteStatement

L'exemple de code suivant montre comment utiliser BatchExecuteStatement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez des lots d'instructions INSERT pour ajouter des éléments.

```
/// <summary>  
/// Inserts movies imported from a JSON file into the movie table by
```

```

    /// using an Amazon DynamoDB PartiQL INSERT statement.
    /// </summary>
    /// <param name="tableName">The name of the table into which the movie
    /// information will be inserted.</param>
    /// <param name="movieFileName">The name of the JSON file that contains
    /// movie information.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the insert operation.</returns>
    public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
    {
        // Get the list of movies from the JSON file.
        var movies = ImportMovies(movieFileName);

        var success = false;

        if (movies is not null)
        {
            // Insert the movies in a batch using PartiQL. Because the
            // batch can contain a maximum of 25 items, insert 25 movies
            // at a time.
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
            var statements = new List<BatchStatementRequest>();

            try
            {
                for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                {
                    for (var i = indexOffset; i < indexOffset + 25; i++)
                    {
                        statements.Add(new BatchStatementRequest
                        {
                            Statement = insertBatch,
                            Parameters = new List<AttributeValue>
                            {
                                new AttributeValue { S = movies[i].Title },
                                new AttributeValue { N =
movies[i].Year.ToString() },
                            },
                        });
                    }
                }
            }
        }
    }

```

```

        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
        {
            Statements = statements,
        });

        // Wait between batches for movies to be successfully added.
        System.Threading.Thread.Sleep(3000);

        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

        // Clear the list of statements for the next batch.
        statements.Clear();
    }
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine(ex.Message);
}
}

return success;
}

/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    if (allMovies is not null)
    {

```

```

        // Return the first 250 entries.
        return allMovies.GetRange(0, 250);
    }
    else
    {
        return null!;
    }
}

```

Utilisez des lots d'instructions SELECT pour obtenir des éléments.

```

/// <summary>
/// Gets movies from the movie table by
/// using an Amazon DynamoDB PartiQL SELECT statement.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year1">The year of the first movie.</param>
/// <param name="year2">The year of the second movie.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> GetBatch(
    string tableName,
    string title1,
    string title2,
    int year1,
    int year2)
{
    var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";

    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title1 },
                new AttributeValue { N = year1.ToString() },
            },
        },
    },
}

```

```

        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title2 },
                new AttributeValue { N = year2.ToString() },
            },
        }
    };

    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
    {
        Statements = statements,
    });

    if (response.Responses.Count > 0)
    {
        response.Responses.ForEach(r =>
        {
            if (r.Item.Any())
            {
                Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
            }
        });
        return true;
    }
    else
    {
        Console.WriteLine($"Couldn't find either {title1} or {title2}.");
        return false;
    }
}

```

Utilisez des lots d'instructions UPDATE pour mettre à jour des éléments.

```

/// <summary>
/// Updates information for multiple movies.
/// </summary>
/// <param name="tableName">The name of the table containing the

```



```

    /// movies to be updated.</param>
    /// <param name="producer1">The producer name for the first movie
    /// to update.</param>
    /// <param name="title1">The title of the first movie.</param>
    /// <param name="year1">The year that the first movie was released.</param>
    /// <param name="producer2">The producer name for the second
    /// movie to update.</param>
    /// <param name="title2">The title of the second movie.</param>
    /// <param name="year2">The year that the second movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the update.</
returns>
    public static async Task<bool> UpdateBatch(
        string tableName,
        string producer1,
        string title1,
        int year1,
        string producer2,
        string title2,
        int year2)
    {
        string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
        var statements = new List<BatchStatementRequest>
        {
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer1 },
                    new AttributeValue { S = title1 },
                    new AttributeValue { N = year1.ToString() },
                },
            },
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer2 },
                    new AttributeValue { S = title2 },
                    new AttributeValue { N = year2.ToString() },
                },
            },
        };
    }
}

```

```

        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Utilisez des lots d'instructions DELETE pour supprimer des éléments.

```

/// <summary>
/// Deletes multiple movies using a PartiQL BatchExecuteAsync
/// statement.
/// </summary>
/// <param name="tableName">The name of the table containing the
/// moves that will be deleted.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="year1">The year the first movie was released.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year2">The year the second movie was released.</param>
/// <returns>A Boolean value indicating the success of the operation.</
returns>
public static async Task<bool> DeleteBatch(
    string tableName,
    string title1,
    int year1,
    string title2,
    int year2)
{
    string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = updateBatch,

```

```
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = title1 },
            new AttributeValue { N = year1.ToString() },
        },
    },

    new BatchStatementRequest
    {
        Statement = updateBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = title2 },
            new AttributeValue { N = year2.ToString() },
        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [BatchExecuteStatement](#) à la section Référence des AWS SDK pour .NET API.

BatchGetItem

L'exemple de code suivant montre comment utiliser `BatchGetItem`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient
client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
                {
                    { _table1Name,
                        new KeysAndAttributes
                        {
                            Keys = new List<Dictionary<string, AttributeValue> >()
                            {
                                new Dictionary<string, AttributeValue>()
                                {
                                    { "Name", new AttributeValue {
                                        S = "Amazon DynamoDB"
                                    } }
                                },
                                new Dictionary<string, AttributeValue>()
                                {
                                    { "Name", new AttributeValue {
                                        S = "Amazon S3"
                                    } }
                                }
                            }
                        }
                    },
                    {
                        _table2Name,
                        new KeysAndAttributes
                        {
                            Keys = new List<Dictionary<string, AttributeValue> >()

```

```

        {
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon DynamoDB"
                } },
                { "Subject", new AttributeValue {
                    S = "DynamoDB Thread 1"
                } }
            },
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon DynamoDB"
                } },
                { "Subject", new AttributeValue {
                    S = "DynamoDB Thread 2"
                } }
            },
            new Dictionary<string, AttributeValue>()
            {
                { "ForumName", new AttributeValue {
                    S = "Amazon S3"
                } },
                { "Subject", new AttributeValue {
                    S = "S3 Thread 1"
                } }
            }
        }
    }
}

BatchGetItemResponse response;
do
{
    Console.WriteLine("Making request");
    response = await client.BatchGetItemAsync(request);

    // Check the response.
    var responses = response.Responses; // Attribute list in the
response.

```

```

        foreach (var tableResponse in responses)
        {
            var tableResults = tableResponse.Value;
            Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
            foreach (var item1 in tableResults)
            {
                PrintItem(item1);
            }
        }

        // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
        Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
        foreach (var unprocessedTableKeys in unprocessedKeys)
        {
            // Print table name.
            Console.WriteLine(unprocessedTableKeys.Key);
            // Print unprocessed primary keys.
            foreach (var key in unprocessedTableKeys.Value.Keys)
            {
                PrintItem(key);
            }
        }

        request.RequestItems = unprocessedKeys;
    } while (response.UnprocessedKeys.Count > 0);
}

private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
{
    foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
    {
        string attributeName = kvp.Key;
        AttributeValue value = kvp.Value;

        Console.WriteLine(
            attributeName + " " +
            (value.S == null ? "" : "S=[" + value.S + "]") +
            (value.N == null ? "" : "N=[" + value.N + "]") +
            (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +

```

```

                (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]")
                );
            }
            Console.WriteLine("*****");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [BatchGetItem](#) à la section Référence des AWS SDK pour .NET API.

DescribeTable

L'exemple de code suivant montre comment utiliser `DescribeTable`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });
}

```

```

        var table = response.Table;
        Console.WriteLine($"Name: {table.TableName}");
        Console.WriteLine($"# of items: {table.ItemCount}");

    }

```

- Pour plus de détails sur l'API, reportez-vous [DescribeTable](#) à la section Référence des AWS SDK pour .NET API.

ExecuteStatement

L'exemple de code suivant montre comment utiliser `ExecuteStatement`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Utilisez une instruction INSERT pour ajouter un élément.

```

    /// <summary>
    /// Inserts a single movie into the movies table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to insert.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the INSERT operation.</returns>
    public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
    {
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {

```



```

        Statement = insertBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
            new AttributeValue { N = year.ToString() },
        },
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

Utilisez une instruction SELECT pour obtenir un élément.

```

/// <summary>
/// Uses a PartiQL SELECT statement to retrieve a single movie from the
/// movie database.
/// </summary>
/// <param name="tableName">The name of the movie table.</param>
/// <param name="movieTitle">The title of the movie to retrieve.</param>
/// <returns>A list of movie data. If no movie matches the supplied
/// title, the list is empty.</returns>
public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
{
    string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
    var parameters = new List<AttributeValue>
    {
        new AttributeValue { S = movieTitle },
    };

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = selectSingle,
        Parameters = parameters,
    });

    return response.Items;
}

```

Utilisez une instruction SELECT pour obtenir une liste d'éléments.

```

    /// <summary>
    /// Retrieve multiple movies by year using a SELECT statement.
    /// </summary>
    /// <param name="tableName">The name of the movie table.</param>
    /// <param name="year">The year the movies were released.</param>
    /// <returns></returns>
    public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { N = year.ToString() },
        };

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = selectSingle,
            Parameters = parameters,
        });

        return response.Items;
    }

```

Utilisez une instruction UPDATE pour mettre à jour un élément.

```

    /// <summary>
    /// Updates a single movie in the table, adding information for the
    /// producer.
    /// </summary>
    /// <param name="tableName">the name of the table.</param>
    /// <param name="producer">The name of the producer.</param>
    /// <param name="movieTitle">The movie title.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the

```

```

    /// UPDATE operation.</returns>
    public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
    {
        string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertSingle,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer },
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

Utilisez une instruction DELETE pour supprimer un seul film.

```

    /// <summary>
    /// Deletes a single movie from the table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to delete.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// DELETE operation.</returns>
    public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
    {
        var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest

```

```
{
    Statement = deleteSingle,
    Parameters = new List<AttributeValue>
    {
        new AttributeValue { S = movieTitle },
        new AttributeValue { N = year.ToString() },
    },
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [ExecuteStatement](#) à la section Référence des AWS SDK pour .NET API.

ListTables

L'exemple de code suivant montre comment utiliser `ListTables`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });
    }
```

```
        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- Pour plus de détails sur l'API, reportez-vous [ListTables](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Créer une application web pour suivre les données DynamoDB

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une table Amazon DynamoDB et envoie des rapports à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser l'API Amazon DynamoDB .NET pour créer une application web dynamique qui suit les données de travail DynamoDB.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Interrogation d'une table à l'aide de lots d'instructions PartiQL

L'exemple de code suivant illustre comment :

- Obtenez un lot d'éléments en exécutant plusieurs instructions SELECT.
- Ajoutez un lot d'éléments en exécutant plusieurs instructions INSERT.
- Mettez à jour un lot d'éléments en exécutant plusieurs instructions UPDATE.
- Supprimez un lot d'éléments en exécutant plusieurs instructions DELETE.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
// Before you run this example, download 'movies.json' from  
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/  
GettingStarted.Js.02.html,
```

```
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\resources\sample_files\movies.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and {title2}.");
```

```
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
    year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
    {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1, year1,
    producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
    year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}
```



```

}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting the
table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.

```

```

/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}

/// <summary>
/// Gets movies from the movie table by
/// using an Amazon DynamoDB PartiQL SELECT statement.
/// </summary>
/// <param name="tableName">The name of the table.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year1">The year of the first movie.</param>
/// <param name="year2">The year of the second movie.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> GetBatch(
    string tableName,
    string title1,
    string title2,
    int year1,
    int year2)
{
    var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";

    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = title1 },
                new AttributeValue { N = year1.ToString() },
            },
        },

        new BatchStatementRequest
        {
            Statement = getBatch,
            Parameters = new List<AttributeValue>

```

```

        {
            new AttributeValue { S = title2 },
            new AttributeValue { N = year2.ToString() },
        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

if (response.Responses.Count > 0)
{
    response.Responses.ForEach(r =>
    {
        if (r.Item.Any())
        {
            Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
        }
    });
    return true;
}
else
{
    Console.WriteLine($"Couldn't find either {title1} or {title2}.");
    return false;
}

}

/// <summary>
/// Inserts movies imported from a JSON file into the movie table by
/// using an Amazon DynamoDB PartiQL INSERT statement.
/// </summary>
/// <param name="tableName">The name of the table into which the movie
/// information will be inserted.</param>
/// <param name="movieFileName">The name of the JSON file that contains
/// movie information.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the insert operation.</returns>
public static async Task<bool> InsertMovies(string tableName, string
movieFileName)

```

```

{
    // Get the list of movies from the JSON file.
    var movies = ImportMovies(movieFileName);

    var success = false;

    if (movies is not null)
    {
        // Insert the movies in a batch using PartiQL. Because the
        // batch can contain a maximum of 25 items, insert 25 movies
        // at a time.
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
        var statements = new List<BatchStatementRequest>();

        try
        {
            for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
            {
                for (var i = indexOffset; i < indexOffset + 25; i++)
                {
                    statements.Add(new BatchStatementRequest
                    {
                        Statement = insertBatch,
                        Parameters = new List<AttributeValue>
                        {
                            new AttributeValue { S = movies[i].Title },
                            new AttributeValue { N =
movies[i].Year.ToString() },
                        },
                    });
                }

                var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                {
                    Statements = statements,
                });

                // Wait between batches for movies to be successfully added.
                System.Threading.Thread.Sleep(3000);

                success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;
            }
        }
    }
}

```

```

        // Clear the list of statements for the next batch.
        statements.Clear();
    }
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine(ex.Message);
}
}

return success;
}

/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The full path to the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public static List<Movie> ImportMovies(string movieFileName)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    if (allMovies is not null)
    {
        // Return the first 250 entries.
        return allMovies.GetRange(0, 250);
    }
    else
    {
        return null!;
    }
}

/// <summary>
/// Updates information for multiple movies.

```

```

    /// </summary>
    /// <param name="tableName">The name of the table containing the
    /// movies to be updated.</param>
    /// <param name="producer1">The producer name for the first movie
    /// to update.</param>
    /// <param name="title1">The title of the first movie.</param>
    /// <param name="year1">The year that the first movie was released.</param>
    /// <param name="producer2">The producer name for the second
    /// movie to update.</param>
    /// <param name="title2">The title of the second movie.</param>
    /// <param name="year2">The year that the second movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the update.</
returns>
    public static async Task<bool> UpdateBatch(
        string tableName,
        string producer1,
        string title1,
        int year1,
        string producer2,
        string title2,
        int year2)
    {

        string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
        var statements = new List<BatchStatementRequest>
        {
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer1 },
                    new AttributeValue { S = title1 },
                    new AttributeValue { N = year1.ToString() },
                },
            },
            new BatchStatementRequest
            {
                Statement = updateBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer2 },

```

```

        new AttributeValue { S = title2 },
        new AttributeValue { N = year2.ToString() },
    },
}
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Deletes multiple movies using a PartiQL BatchExecuteAsync
/// statement.
/// </summary>
/// <param name="tableName">The name of the table containing the
/// moves that will be deleted.</param>
/// <param name="title1">The title of the first movie.</param>
/// <param name="year1">The year the first movie was released.</param>
/// <param name="title2">The title of the second movie.</param>
/// <param name="year2">The year the second movie was released.</param>
/// <returns>A Boolean value indicating the success of the operation.</
returns>
public static async Task<bool> DeleteBatch(
    string tableName,
    string title1,
    int year1,
    string title2,
    int year2)
{
    string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

    var statements = new List<BatchStatementRequest>
    {
        new BatchStatementRequest
        {
            Statement = updateBatch,
            Parameters = new List<AttributeValue>
            {

```

```
        new AttributeValue { S = title1 },
        new AttributeValue { N = year1.ToString() },
    },
    new BatchStatementRequest
    {
        Statement = updateBatch,
        Parameters = new List<AttributeValue>
        {
            new AttributeValue { S = title2 },
            new AttributeValue { N = year2.ToString() },
        },
    }
};

var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
{
    Statements = statements,
});

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [BatchExecuteStatement](#) à la section Référence des AWS SDK pour .NET API.

Interrogation d'une table à l'aide de PartiQL

L'exemple de code suivant illustre comment :

- Obtenez un élément en exécutant une instruction SELECT.
- Ajoutez un élément en exécutant une instruction INSERT.
- Mettez à jour un élément en exécutant une instruction UPDATE.
- Supprimez un élément en exécutant une instruction DELETE.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
AmazonDynamoDBClient();

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();
```

```

        try
        {
            for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
            {
                for (var i = indexOffset; i < indexOffset + 25; i++)
                {
                    statements.Add(new BatchStatementRequest
                    {
                        Statement = insertBatch,
                        Parameters = new List<AttributeValue>
                        {
                            new AttributeValue { S = movies[i].Title },
                            new AttributeValue { N =
movies[i].Year.ToString() },
                        },
                    });
                }

                var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                {
                    Statements = statements,
                });

                // Wait between batches for movies to be successfully added.
                System.Threading.Thread.Sleep(3000);

                success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                // Clear the list of statements for the next batch.
                statements.Clear();
            }
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    return success;
}

```

```

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public static List<Movie> ImportMovies(string movieFileName)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        if (allMovies is not null)
        {
            // Return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }
        else
        {
            return null!;
        }
    }

    /// <summary>
    /// Uses a PartiQL SELECT statement to retrieve a single movie from the
    /// movie database.
    /// </summary>
    /// <param name="tableName">The name of the movie table.</param>
    /// <param name="movieTitle">The title of the movie to retrieve.</param>
    /// <returns>A list of movie data. If no movie matches the supplied
    /// title, the list is empty.</returns>
    public static async Task<List<Dictionary<string, AttributeValue>>>
    GetSingleMovie(string tableName, string movieTitle)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
        var parameters = new List<AttributeValue>
        {

```

```

        new AttributeValue { S = movieTitle },
    };

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = selectSingle,
        Parameters = parameters,
    });

    return response.Items;
}

/// <summary>
/// Retrieve multiple movies by year using a SELECT statement.
/// </summary>
/// <param name="tableName">The name of the movie table.</param>
/// <param name="year">The year the movies were released.</param>
/// <returns></returns>
public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
{
    string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
    var parameters = new List<AttributeValue>
    {
        new AttributeValue { N = year.ToString() },
    };

    var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
    {
        Statement = selectSingle,
        Parameters = parameters,
    });

    return response.Items;
}

/// <summary>
/// Inserts a single movie into the movies table.
/// </summary>

```

```

    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to insert.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the INSERT operation.</returns>
    public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
    {
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Updates a single movie in the table, adding information for the
    /// producer.
    /// </summary>
    /// <param name="tableName">the name of the table.</param>
    /// <param name="producer">The name of the producer.</param>
    /// <param name="movieTitle">The movie title.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// UPDATE operation.</returns>
    public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
    {
        string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

```

```

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertSingle,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer },
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Deletes a single movie from the table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to delete.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// DELETE operation.</returns>
    public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
    {
        var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = deleteSingle,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

```

    /// <summary>
    /// Displays the list of movies returned from a database query.
    /// </summary>
    /// <param name="items">The list of movie information to display.</param>
    private static void DisplayMovies(List<Dictionary<string, AttributeValue>>
items)
    {
        if (items.Count > 0)
        {
            Console.WriteLine($"Found {items.Count} movies.");
            items.ForEach(item =>
Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
        }
        else
        {
            Console.WriteLine($"Didn't find a movie that matched the supplied
criteria.");
        }
    }
}

}

}

    /// <summary>
    /// Uses a PartiQL SELECT statement to retrieve a single movie from the
    /// movie database.
    /// </summary>
    /// <param name="tableName">The name of the movie table.</param>
    /// <param name="movieTitle">The title of the movie to retrieve.</param>
    /// <returns>A list of movie data. If no movie matches the supplied
    /// title, the list is empty.</returns>
    public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
    {
        string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
        var parameters = new List<AttributeValue>
        {
            new AttributeValue { S = movieTitle },
        };
    }
}

```

```

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = selectSingle,
            Parameters = parameters,
        });

        return response.Items;
    }

    /// <summary>
    /// Inserts a single movie into the movies table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to insert.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the INSERT operation.</returns>
    public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
    {
        string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertBatch,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>

```



```

    /// Updates a single movie in the table, adding information for the
    /// producer.
    /// </summary>
    /// <param name="tableName">the name of the table.</param>
    /// <param name="producer">The name of the producer.</param>
    /// <param name="movieTitle">The movie title.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// UPDATE operation.</returns>
    public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
    {
        string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

        var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
        {
            Statement = insertSingle,
            Parameters = new List<AttributeValue>
            {
                new AttributeValue { S = producer },
                new AttributeValue { S = movieTitle },
                new AttributeValue { N = year.ToString() },
            },
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Deletes a single movie from the table.
    /// </summary>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="movieTitle">The title of the movie to delete.</param>
    /// <param name="year">The year that the movie was released.</param>
    /// <returns>A Boolean value that indicates the success of the
    /// DELETE operation.</returns>
    public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
    {

```

```
var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year  
= ?";  
  
var response = await Client.ExecuteStatementAsync(new  
ExecuteStatementRequest  
{  
    Statement = deleteSingle,  
    Parameters = new List<AttributeValue>  
    {  
        new AttributeValue { S = movieTitle },  
        new AttributeValue { N = year.ToString() },  
    },  
});  
  
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
}
```

- Pour plus de détails sur l'API, reportez-vous [ExecuteStatement](#) à la section Référence des AWS SDK pour .NET API.

Utiliser un modèle de document

L'exemple de code suivant montre comment effectuer des opérations de création, de lecture, de mise à jour et de suppression (CRUD) et par lots à l'aide d'un modèle de document pour DynamoDB et d'un SDK. AWS

Pour en savoir plus, consultez la section [Modèle de document](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Effectuez des opérations CRUD à l'aide d'un modèle de document.

```
/// <summary>
```

```

/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
    /// <param name="tableName">The name of the table to load.</param>
    /// <returns>A DynamoDB table object.</returns>
    public static Table LoadTable(IAmazonDynamoDB client, string tableName)
    {
        Table productCatalog = Table.LoadTable(client, tableName);
        return productCatalog;
    }

    /// <summary>
    /// Creates an example book item and adds it to the DynamoDB table
    /// ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)

```

```

    {
        Console.WriteLine("\n*** Executing CreateBookItem() ***");
        var book = new Document
        {
            ["Id"] = sampleBookId,
            ["Title"] = "Book " + sampleBookId,
            ["Price"] = 19.99,
            ["ISBN"] = "111-1111111111",
            ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
            ["PageCount"] = 500,
            ["Dimensions"] = "8.5x11x.5",
            ["InPublication"] = new DynamoDBBool(true),
            ["InStock"] = new DynamoDBBool(false),
            ["QuantityOnHand"] = 0,
        };

        // Adds the book to the ProductCatalog table.
        await productCatalog.PutItemAsync(book);
    }

    /// <summary>
    /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void RetrieveBook(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing RetrieveBook() ***");

        // Optional configuration.
        var config = new GetItemOperationConfig
        {
            AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
            ConsistentRead = true,
        };

        Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
        Console.WriteLine("RetrieveBook: Printing book retrieved...");
    }

```

```

        PrintDocument(document);
    }

    /// <summary>
    /// Updates multiple attributes for a book and writes the changes to the
    /// DynamoDB table ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void UpdateMultipleAttributes(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\nUpdating multiple attributes....");
        int partitionKey = sampleBookId;

        var book = new Document
        {
            ["Id"] = partitionKey,

            // List of attribute updates.
            // The following replaces the existing authors list.
            ["Authors"] = new List<string> { "Author x", "Author y" },
            ["newAttribute"] = "New Value",
            ["ISBN"] = null, // Remove it.
        };

        // Optional parameters.
        var config = new UpdateItemOperationConfig
        {
            // Gets updated item in response.
            ReturnValues = ReturnValues.AllNewAttributes,
        };

        Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
        Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
        PrintDocument(updatedBook);
    }

    /// <summary>
    /// Updates a book item if it meets the specified criteria.

```

```

    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>
    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async void UpdateBookPriceConditionally(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

        int partitionKey = sampleBookId;

        var book = new Document
        {
            ["Id"] = partitionKey,
            ["Price"] = 29.99,
        };

        // For conditional price update, creating a condition expression.
        var expr = new Expression
        {
            ExpressionStatement = "Price = :val",
        };
        expr.ExpressionAttributeValues[":val"] = 19.00;

        // Optional parameters.
        var config = new UpdateItemOperationConfig
        {
            ConditionalExpression = expr,
            ReturnValues = ReturnValues.AllNewAttributes,
        };

        Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
        Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
        PrintDocument(updatedBook);
    }

    /// <summary>
    /// Deletes the book with the supplied Id value from the DynamoDB table
    /// ProductCatalog.
    /// </summary>
    /// <param name="productCatalog">A DynamoDB table object.</param>

```

```

    /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
    public static async Task DeleteBook(
        Table productCatalog,
        int sampleBookId)
    {
        Console.WriteLine("\n*** Executing DeleteBook() ***");

        // Optional configuration.
        var config = new DeleteItemOperationConfig
        {
            // Returns the deleted item.
            ReturnValues = ReturnValues.AllOldAttributes,
        };
        Document document = await productCatalog.DeleteItemAsync(sampleBookId,
config);
        Console.WriteLine("DeleteBook: Printing deleted just deleted...");

        PrintDocument(document);
    }

    /// <summary>
    /// Prints the information for the supplied DynamoDB document.
    /// </summary>
    /// <param name="updatedDocument">A DynamoDB document object.</param>
    public static void PrintDocument(Document updatedDocument)
    {
        if (updatedDocument is null)
        {
            return;
        }

        foreach (var attribute in updatedDocument.GetAttributeNames())
        {
            string stringValue = null;
            var value = updatedDocument[attribute];

            if (value is null)
            {
                continue;
            }

            if (value is Primitive)
            {

```

```

        stringValue = value.AsPrimitive().Value.ToString();
    }
    else if (value is PrimitiveList)
    {
        stringValue = string.Join(",", (from primitive
                                         in value.AsPrimitiveList().Entries
                                         select
primitive.Value).ToArray());
    }

    Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
    }
}
}

```

Effectuez des opérations d'écriture par lots à l'aide d'un modèle de document.

```

/// <summary>
/// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
/// operations.
/// </summary>
public class MidLevelBatchWriteItem
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();

        await SingleTableBatchWrite(client);
        await MultiTableBatchWrite(client);
    }

    /// <summary>
    /// Perform a batch operation on a single DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB object.</param>
    public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
    {
        Table productCatalog = Table.LoadTable(client, "ProductCatalog");
        var batchWrite = productCatalog.CreateBatchWrite();
    }
}

```



```

var book1 = new Document
{
    ["Id"] = 902,
    ["Title"] = "My book1 in batch write using .NET helper classes",
    ["ISBN"] = "902-11-11-1111",
    ["Price"] = 10,
    ["ProductCategory"] = "Book",
    ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
    ["Dimensions"] = "8.5x11x.5",
    ["InStock"] = new DynamoDBBool(true),
    ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown at
this time.
};

batchWrite.AddDocumentToPut(book1);

// Specify delete item using overload that takes PK.
batchWrite.AddKeyToDelete(12345);
Console.WriteLine("Performing batch write in SingleTableBatchWrite()");
await batchWrite.ExecuteAsync();
}

/// <summary>
/// Perform a batch operation involving multiple DynamoDB tables.
/// </summary>
/// <param name="client">An initialized DynamoDB client object.</param>
public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
{
    // Specify item to add in the Forum table.
    Table forum = Table.LoadTable(client, "Forum");
    var forumBatchWrite = forum.CreateBatchWrite();

    var forum1 = new Document
    {
        ["Name"] = "Test BatchWrite Forum",
        ["Threads"] = 0,
    };
    forumBatchWrite.AddDocumentToPut(forum1);

    // Specify item to add in the Thread table.
    Table thread = Table.LoadTable(client, "Thread");
    var threadBatchWrite = thread.CreateBatchWrite();

```

```

var thread1 = new Document
{
    ["ForumName"] = "S3 forum",
    ["Subject"] = "My sample question",
    ["Message"] = "Message text",
    ["KeywordTags"] = new List<string> { "S3", "Bucket" },
};
threadBatchWrite.AddDocumentToPut(thread1);

// Specify item to delete from the Thread table.
threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

// Create multi-table batch.
var superBatch = new MultiTableDocumentBatchWrite();
superBatch.AddBatch(forumBatchWrite);
superBatch.AddBatch(threadBatchWrite);
Console.WriteLine("Performing batch write in MultiTableBatchWrite()");

// Execute the batch.
await superBatch.ExecuteAsync();
}
}

```

Analysez une table à l'aide d'un modèle de document.

```

/// <summary>
/// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
/// table for values.
/// </summary>
public class MidLevelScanOnly
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();

        Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");

        await FindProductsWithNegativePrice(productCatalogTable);
        await FindProductsWithNegativePriceWithConfig(productCatalogTable);
    }
}

```

```

    }

    /// <summary>
    /// Retrieves any products that have a negative price in a DynamoDB table.
    /// </summary>
    /// <param name="productCatalogTable">A DynamoDB table object.</param>
    public static async Task FindProductsWithNegativePrice(
        Table productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        var scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

        Search search = productCatalogTable.Scan(scanFilter);

        do
        {
            var documentList = await search.GetNextSetAsync();
            Console.WriteLine("\nFindProductsWithNegativePrice:
printing .....");

            foreach (var document in documentList)
            {
                PrintDocument(document);
            }
        } while (!search.IsDone);
    }

    /// <summary>
    /// Finds any items in the ProductCatalog table using a DynamoDB
    /// configuration object.
    /// </summary>
    /// <param name="productCatalogTable">A DynamoDB table object.</param>
    public static async Task FindProductsWithNegativePriceWithConfig(
        Table productCatalogTable)
    {
        // Assume there is a price error. So we scan to find items priced < 0.
        var scanFilter = new ScanFilter();
        scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

        var config = new ScanOperationConfig()
        {
            Filter = scanFilter,

```

```

        Select = SelectValues.SpecificAttributes,
        AttributesToGet = new List<string> { "Title", "Id" },
    };

    Search search = productCatalogTable.Scan(config);

    do
    {
        var documentList = await search.GetNextSetAsync();
        Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

        foreach (var document in documentList)
        {
            PrintDocument(document);
        }
    }
    while (!search.IsDone);
}

/// <summary>
/// Displays the details of the passed DynamoDB document object on the
/// console.
/// </summary>
/// <param name="document">A DynamoDB document object.</param>
public static void PrintDocument(Document document)
{
    Console.WriteLine();
    foreach (var attribute in document.GetAttributeNames())
    {
        string stringValue = null;
        var value = document[attribute];
        if (value is Primitive)
        {
            stringValue = value.AsPrimitive().Value.ToString();
        }
        else if (value is PrimitiveList)
        {
            stringValue = string.Join(",", (from primitive
                                             in value.AsPrimitiveList().Entries
                                             select
primitive.Value).ToArray());
        }
    }
}

```

```

        Console.WriteLine($"{attribute} - {stringValue}");
    }
}
}

```

Interrogez et analysez une table à l'aide d'un modèle de document.

```

/// <summary>
/// Shows how to perform mid-level query procedures on an Amazon DynamoDB
/// table.
/// </summary>
public class MidLevelQueryAndScan
{
    public static async Task Main()
    {
        IAmazonDynamoDB client = new AmazonDynamoDBClient();

        // Query examples.
        Table replyTable = Table.LoadTable(client, "Reply");
        string forumName = "Amazon DynamoDB";
        string threadSubject = "DynamoDB Thread 2";

        await FindRepliesInLast15Days(replyTable);
        await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
        await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

        // Get Example.
        Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
        int productId = 101;

        await GetProduct(productCatalogTable, productId);
    }

    /// <summary>
    /// Retrieves information about a product from the DynamoDB table
    /// ProductCatalog based on the product ID and displays the information
    /// on the console.
    /// </summary>

```

```

    /// <param name="tableName">The name of the table from which to retrieve
    /// product information.</param>
    /// <param name="productId">The ID of the product to retrieve.</param>
    public static async Task GetProduct(Table tableName, int productId)
    {
        Console.WriteLine("*** Executing GetProduct() ***");
        Document productDocument = await tableName.GetItemAsync(productId);
        if (productDocument != null)
        {
            PrintDocument(productDocument);
        }
        else
        {
            Console.WriteLine("Error: product " + productId + " does not
exist");
        }
    }

    /// <summary>
    /// Retrieves replies from the passed DynamoDB table object.
    /// </summary>
    /// <param name="table">The table we want to query.</param>
    public static async Task FindRepliesInLast15Days(
        Table table)
    {
        DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
        var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
        filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

        // Use Query overloads that take the minimum required query parameters.
        Search search = table.Query(filter);

        do
        {
            var documentSet = await search.GetNextSetAsync();
            Console.WriteLine("\nFindRepliesInLast15Days:
printing .....");

            foreach (var document in documentSet)
            {
                PrintDocument(document);
            }
        }
    }

```

```

        while (!search.IsDone);
    }

    /// <summary>
    /// Retrieve replies made during a specific time period.
    /// </summary>
    /// <param name="table">The table we want to query.</param>
    /// <param name="forumName">The name of the forum that we're interested
in.</param>
    /// <param name="threadSubject">The subject of the thread, which we are
    /// searching for replies.</param>
    public static async Task FindRepliesPostedWithinTimePeriod(
        Table table,
        string forumName,
        string threadSubject)
    {
        DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
        DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0, 0));

        var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadSubject);
        filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

        var config = new QueryOperationConfig()
        {
            Limit = 2, // 2 items/page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
        {
            "Message",
            "ReplyDateTime",
            "PostedBy",
        },
            ConsistentRead = true,
            Filter = filter,
        };

        Search search = table.Query(config);

        do
        {
            var documentList = await search.GetNextSetAsync();

```

```

        Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

        foreach (var document in documentList)
        {
            PrintDocument(document);
        }
    }
    while (!search.IsDone);
}

/// <summary>
/// Perform a query for replies made in the last 15 days using a DynamoDB
/// QueryOperationConfig object.
/// </summary>
/// <param name="table">The table we want to query.</param>
/// <param name="forumName">The name of the forum that we're interested
in.</param>
/// <param name="threadName">The name of the thread that we are searching
/// for replies.</param>
public static async Task FindRepliesInLast15DaysWithConfig(
    Table table,
    string forumName,
    string threadName)
{
    DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
    var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadName);
    filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

    var config = new QueryOperationConfig()
    {
        Filter = filter,

        // Optional parameters.
        Select = SelectValues.SpecificAttributes,
        AttributesToGet = new List<string>
        {
            "Message",
            "ReplyDateTime",
            "PostedBy",
        },
        ConsistentRead = true,
    }
}

```



```

    };

    Search search = table.Query(config);

    do
    {
        var documentSet = await search.GetNextSetAsync();
        Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

        foreach (var document in documentSet)
        {
            PrintDocument(document);
        }
    }
    while (!search.IsDone);
}

/// <summary>
/// Displays the contents of the passed DynamoDB document on the console.
/// </summary>
/// <param name="document">A DynamoDB document to display.</param>
public static void PrintDocument(Document document)
{
    Console.WriteLine();
    foreach (var attribute in document.GetAttributeNames())
    {
        string stringValue = null;
        var value = document[attribute];

        if (value is Primitive)
        {
            stringValue = value.AsPrimitive().Value.ToString();
        }
        else if (value is PrimitiveList)
        {
            stringValue = string.Join(",", (from primitive
                                             in value.AsPrimitiveList().Entries
                                             select
primitive.Value).ToArray());
        }

        Console.WriteLine($"{attribute} - {stringValue}");
    }
}

```

```
}  
}
```

Utiliser un modèle de persistance des objets de haut niveau

L'exemple de code suivant montre comment effectuer des opérations de création, de lecture, de mise à jour et de suppression (CRUD) et par lots à l'aide d'un modèle de persistance d'objets pour DynamoDB et d'un SDK. AWS

Pour plus d'informations, consultez la section [Modèle de persistance des objets](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Effectuez des opérations CRUD à l'aide d'un modèle de persistance des objets de haut niveau.

```
/// <summary>  
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB  
/// table.  
/// </summary>  
public class HighLevelItemCrud  
{  
    public static async Task Main()  
    {  
        var client = new AmazonDynamoDBClient();  
        DynamoDBContext context = new DynamoDBContext(client);  
        await PerformCRUDOperations(context);  
    }  
  
    public static async Task PerformCRUDOperations(IDynamoDBContext context)  
    {  
        int bookId = 1001; // Some unique value.  
        Book myBook = new Book
```

```

    {
        Id = bookId,
        Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
        Isbn = "111-1111111001",
        BookAuthors = new List<string> { "Author 1", "Author 2" },
    };

    // Save the book to the ProductCatalog table.
    await context.SaveChangesAsync();

    // Retrieve the book from the ProductCatalog table.
    Book bookRetrieved = await context.LoadAsync<Book>(bookId);

    // Update some properties.
    bookRetrieved.Isbn = "222-2222221001";

    // Update existing authors list with the following values.
    bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };

    await context.SaveChangesAsync();

    // Retrieve the updated book. This time, add the optional
    // ConsistentRead parameter using DynamoDBContextConfig object.
    await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
    {
        ConsistentRead = true,
    });

    // Delete the book.
    await context.DeleteAsync<Book>(bookId);

    // Try to retrieve deleted book. It should return null.
    Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
    {
        ConsistentRead = true,
    });

    if (deletedBook == null)
    {
        Console.WriteLine("Book is deleted");
    }
}
}

```

Effectuez des opérations d'écriture par lots à l'aide d'un modèle de persistance des objets de haut niveau.

```
/// <summary>
/// Performs high-level batch write operations to an Amazon DynamoDB table.
/// This example was written using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class HighLevelBatchWriteItem
{
    public static async Task SingleTableBatchWrite(IDynamoDBContext context)
    {
        Book book1 = new Book
        {
            Id = 902,
            InPublication = true,
            Isbn = "902-11-11-1111",
            PageCount = "100",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book3 in batch write",
        };

        Book book2 = new Book
        {
            Id = 903,
            InPublication = true,
            Isbn = "903-11-11-1111",
            PageCount = "200",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book4 in batch write",
        };

        var bookBatch = context.CreateBatchWrite<Book>();
        bookBatch.AddPutItems(new List<Book> { book1, book2 });

        Console.WriteLine("Adding two books to ProductCatalog table.");
        await bookBatch.ExecuteAsync();
    }
}
```

```

    }

    public static async Task MultiTableBatchWrite(IDynamoDBContext context)
    {
        // New Forum item.
        Forum newForum = new Forum
        {
            Name = "Test BatchWrite Forum",
            Threads = 0,
        };
        var forumBatch = context.CreateBatchWrite<Forum>();
        forumBatch.AddPutItem(newForum);

        // New Thread item.
        Thread newThread = new Thread
        {
            ForumName = "S3 forum",
            Subject = "My sample question",
            KeywordTags = new List<string> { "S3", "Bucket" },
            Message = "Message text",
        };

        DynamoDBOperationConfig config = new DynamoDBOperationConfig();
        config.SkipVersionCheck = true;
        var threadBatch = context.CreateBatchWrite<Thread>(config);
        threadBatch.AddPutItem(newThread);
        threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

        var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

        Console.WriteLine("Performing batch write in MultiTableBatchWrite().");
        await superBatch.ExecuteAsync();
    }

    public static async Task Main()
    {
        AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);

        await SingleTableBatchWrite(context);
        await MultiTableBatchWrite(context);
    }
}

```

Mappez des données arbitraires à une table à l'aide d'un modèle de persistance des objets de haut niveau.

```
/// <summary>
/// Shows how to map arbitrary data to an Amazon DynamoDB table.
/// </summary>
public class HighLevelMappingArbitraryData
{
    /// <summary>
    /// Creates a book, adds it to the DynamoDB ProductCatalog table, retrieves
    /// the new book from the table, updates the dimensions and writes the
    /// changed item back to the table.
    /// </summary>
    /// <param name="context">The DynamoDB context object used to write and
    /// read data from the table.</param>
    public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
    {
        // Create a book.
        DimensionType myBookDimensions = new DimensionType()
        {
            Length = 8M,
            Height = 11M,
            Thickness = 0.5M,
        };

        Book myBook = new Book
        {
            Id = 501,
            Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
            Isbn = "999-9999999999",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
            Dimensions = myBookDimensions,
        };

        // Add the book to the DynamoDB table ProductCatalog.
        await context.SaveAsync(myBook);

        // Retrieve the book.
```

```

        Book bookRetrieved = await context.LoadAsync<Book>(501);

        // Update the book dimensions property.
        bookRetrieved.Dimensions.Height += 1;
        bookRetrieved.Dimensions.Length += 1;
        bookRetrieved.Dimensions.Thickness += 0.2M;

        // Write the changed item to the table.
        await context.SaveAsync(bookRetrieved);
    }

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await AddRetrieveUpdateBook(context);
    }
}

```

Interrogez et analysez une table à l'aide d'un modèle de persistance des objets de haut niveau.

```

/// <summary>
/// Shows how to perform high-level query and scan operations to Amazon
/// DynamoDB tables.
/// </summary>
public class HighLevelQueryAndScan
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        DynamoDBContext context = new DynamoDBContext(client);

        // Get an item.
        await GetBook(context, 101);

        // Sample forum and thread to test queries.
        string forumName = "Amazon DynamoDB";
        string threadSubject = "DynamoDB Thread 1";
    }
}

```

```

        // Sample queries.
        await FindRepliesInLast15Days(context, forumName, threadSubject);
        await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

        // Scan table.
        await FindProductsPricedLessThanZero(context);
    }

    public static async Task GetBook(IDynamoDBContext context, int productId)
    {
        Book bookItem = await context.LoadAsync<Book>(productId);

        Console.WriteLine("\nGetBook: Printing result.....");
        Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn} \n
No. of pages: {bookItem.PageCount}");
    }

    /// <summary>
    /// Queries a DynamoDB table to find replies posted within the last 15 days.
    /// </summary>
    /// <param name="context">The DynamoDB context used to perform the query.</
param>
    /// <param name="forumName">The name of the forum that we're interested
in.</param>
    /// <param name="threadSubject">The thread object containing the query
parameters.</param>
    public static async Task FindRepliesInLast15Days(
        IDynamoDBContext context,
        string forumName,
        string threadSubject)
    {
        string replyId = $"{forumName} #{threadSubject}";
        DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

        List<object> times = new List<object>();
        times.Add(twoWeeksAgoDate);

        List<ScanCondition> scs = new List<ScanCondition>();
        var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
        scs.Add(sc);

        var cfg = new DynamoDBOperationConfig

```



```

        {
            QueryFilter = scs,
        };

        AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId, cfg);
        IEnumerable<Reply> latestReplies = await response.GetRemainingAsync();

        Console.WriteLine("\nReplies in last 15 days:");

        foreach (Reply r in latestReplies)
        {
            Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
        }
    }

    /// <summary>
    /// Queries for replies posted within a specific time period.
    /// </summary>
    /// <param name="context">The DynamoDB context used to perform the query.</
param>
    /// <param name="forumName">The name of the forum that we're interested
in.</param>
    /// <param name="threadSubject">Information about the subject that we're
    /// interested in.</param>
    public static async Task FindRepliesPostedWithinTimePeriod(
        IDynamoDBContext context,
        string forumName,
        string threadSubject)
    {
        string forumId = forumName + "#" + threadSubject;
        Console.WriteLine("\nReplies posted within time period:");

        DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
        DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

        List<object> times = new List<object>();
        times.Add(startDate);
        times.Add(endDate);

        List<ScanCondition> scs = new List<ScanCondition>();
        var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
        scs.Add(sc);
    }

```

```

        var cfg = new DynamoDBOperationConfig
        {
            QueryFilter = scs,
        };

        AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId, cfg);
        IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

        foreach (Reply r in repliesInAPeriod)
        {
            Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
        }
    }

    /// <summary>
    /// Queries the DynamoDB ProductCatalog table for products costing less
    /// than zero.
    /// </summary>
    /// <param name="context">The DynamoDB context object used to perform the
    /// query.</param>
    public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
    {
        int price = 0;

        List<ScanCondition> scs = new List<ScanCondition>();
        var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
        var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
        scs.Add(sc1);
        scs.Add(sc2);

        AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

        IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

        Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

        foreach (Book r in itemsWithWrongPrice)

```

```
        {  
            Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");  
        }  
    }  
}
```

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux DynamoDB. La fonction récupère les données utiles DynamoDB et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement DynamoDB avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
using System.Text.Json;  
using System.Text;  
using Amazon.Lambda.Core;  
using Amazon.Lambda.DynamoDBEvents;  
  
// Assembly attribute to enable the Lambda function's JSON input to be converted  
// into a .NET class.  
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))  
]  
  
namespace AWSLambda_DDB;  
  
public class Function
```

```
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
{dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux DynamoDB. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
```

```

using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}

```

```
}  
}
```

AWS contributions communautaires

Création et test d'une application sans serveur

L'exemple de code suivant montre comment créer et tester une application sans serveur à l'aide d'API Gateway avec Lambda et DynamoDB.

SDK pour .NET

Illustre comment créer et tester une application sans serveur composée d'une API Gateway avec Lambda et DynamoDB à l'aide du kit SDK .NET.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

EC2 Exemples d'Amazon utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon EC2.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Principes de base](#)
- [Actions](#)
- [Scénarios](#)

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une paire de clés et un groupe de sécurité.
- Sélectionnez une Amazon Machine Image (AMI) et un type d'instance compatible, puis créez une instance.
- Arrêtez l'instance, puis redémarrez-la.
- Associez une adresse IP Elastic à votre instance
- Connectez-vous à votre instance avec SSH, puis nettoyez les ressources.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario à une invite de commande.

```
/// <summary>
/// Show Amazon Elastic Compute Cloud (Amazon EC2) Basics actions.
/// </summary>
public class EC2Basics
{
    public static ILogger<EC2Basics> _logger = null!;
    public static EC2Wrapper _ec2Wrapper = null!;
```

```

public static SsmWrapper _ssmWrapper = null!;
public static UiMethods _uiMethods = null!;

public static string associationId = null!;
public static string allocationId = null!;
public static string instanceId = null!;
public static string keyPairName = null!;
public static string groupName = null!;
public static string tempFileName = null!;
public static string secGroupId = null!;
public static bool isInteractive = true;

/// <summary>
/// Perform the actions defined for the Amazon EC2 Basics scenario.
/// </summary>
/// <param name="args">Command line arguments.</param>
/// <returns>A Task object.</returns>
public static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EC2 and Amazon Simple Systems
    // Management (Amazon SSM) Service.
    using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonEC2>()
            .AddAWSService<IAmazonSimpleSystemsManagement>()
            .AddTransient<EC2Wrapper>()
            .AddTransient<SsmWrapper>()
        )
    .Build();

    SetUpServices(host);

    var uniqueName = Guid.NewGuid().ToString();
    keyPairName = "mvp-example-key-pair" + uniqueName;
    groupName = "ec2-scenario-group" + uniqueName;
    var groupDescription = "A security group created for the EC2 Basics
scenario.";

    try
    {
        // Start the scenario.
        _uiMethods.DisplayOverview();
        _uiMethods.PressEnter(isInteractive);
    }
}

```



```

// Create the key pair.
_uiMethods.DisplayTitle("Create RSA key pair");
Console.Write("Let's create an RSA key pair that you can be use to ");
Console.WriteLine("securely connect to your EC2 instance.");
var keyPair = await _ec2Wrapper.CreateKeyPair(keyPairName);

// Save key pair information to a temporary file.
tempFileName = _ec2Wrapper.SaveKeyPair(keyPair);

Console.WriteLine(
    $"Created the key pair: {keyPair.KeyName} and saved it to:
{tempFileName}");
string? answer = "";
if (isInteractive)
{
    do
    {
        Console.Write("Would you like to list your existing key pairs?
");
        answer = Console.ReadLine();
    } while (answer!.ToLower() != "y" && answer.ToLower() != "n");
}

if (!isInteractive || answer == "y")
{
    // List existing key pairs.
    _uiMethods.DisplayTitle("Existing key pairs");

    // Passing an empty string to the DescribeKeyPairs method will
return
    // a list of all existing key pairs.
    var keyPairs = await _ec2Wrapper.DescribeKeyPairs("");
    keyPairs.ForEach(kp =>
    {
        Console.WriteLine(
            $"{kp.KeyName} created at: {kp.CreateTime} Fingerprint:
{kp.KeyFingerprint}");
    });
}

_uiMethods.PressEnter(isInteractive);

// Create the security group.

```

```

        Console.WriteLine(
            "Let's create a security group to manage access to your instance.");
        secGroupId = await _ec2Wrapper.CreateSecurityGroup(groupName,
groupDescription);
        Console.WriteLine(
            "Let's add rules to allow all HTTP and HTTPS inbound traffic and to
allow SSH only from your current IP address.");

        _uiMethods.DisplayTitle("Security group information");
        var secGroups = await _ec2Wrapper.DescribeSecurityGroups(secGroupId);

        Console.WriteLine($"Created security group {groupName} in your default
VPC.");
        secGroups.ForEach(group =>
        {
            _ec2Wrapper.DisplaySecurityGroupInfoAsync(group);
        });
        _uiMethods.PressEnter(isInteractive);

        Console.WriteLine(
            "Now we'll authorize the security group we just created so that it
can");
        Console.WriteLine("access the EC2 instances you create.");
        await _ec2Wrapper.AuthorizeSecurityGroupIngress(groupName);

        secGroups = await _ec2Wrapper.DescribeSecurityGroups(secGroupId);
        Console.WriteLine($"Now let's look at the permissions again.");
        secGroups.ForEach(group =>
        {
            _ec2Wrapper.DisplaySecurityGroupInfoAsync(group);
        });
        _uiMethods.PressEnter(isInteractive);

        // Get list of available Amazon Linux 2 Amazon Machine Images (AMIs).
        var parameters =
            await _ssmWrapper.GetParametersByPath(
                "/aws/service/ami-amazon-linux-latest");

        List<string> imageIds = parameters.Select(param =>
param.Value).ToList();

        var images = await _ec2Wrapper.DescribeImages(imageIds);

        var i = 1;

```

```
images.ForEach(image =>
{
    Console.WriteLine($"{i++}\t{image.Description}");
});

int choice = 1;
bool validNumber = false;
if (isInteractive)
{
    do
    {
        Console.Write("Please select an image: ");
        var selImage = Console.ReadLine();
        validNumber = int.TryParse(selImage, out choice);
    } while (!validNumber);
}

var selectedImage = images[choice - 1];

// Display available instance types.
_uiMethods.DisplayTitle("Instance Types");
var instanceTypes =
    await _ec2Wrapper.DescribeInstanceTypes(selectedImage.Architecture);

i = 1;
instanceTypes.ForEach(instanceType =>
{
    Console.WriteLine($"{i++}\t{instanceType.InstanceType}");
});
if (isInteractive)
{
    do
    {
        Console.Write("Please select an instance type: ");
        var selImage = Console.ReadLine();
        validNumber = int.TryParse(selImage, out choice);
    } while (!validNumber);
}

var selectedInstanceType = instanceTypes[choice - 1].InstanceType;

// Create an EC2 instance.
_uiMethods.DisplayTitle("Creating an EC2 Instance");
instanceId = await _ec2Wrapper.RunInstances(selectedImage.ImageId,
```

```
        selectedInstanceType, keyPairName, secGroupId);

        _uiMethods.PressEnter(isInteractive);

        var instance = await _ec2Wrapper.DescribeInstance(instanceId);
        _uiMethods.DisplayTitle("New Instance Information");
        _ec2Wrapper.DisplayInstanceInformation(instance);

        Console.WriteLine(
            "\nYou can use SSH to connect to your instance. For example:");
        Console.WriteLine(
            $"\\tssh -i {tempFileName} ec2-user@{instance.PublicIpAddress}");

        _uiMethods.PressEnter(isInteractive);

        Console.WriteLine(
            "Now we'll stop the instance and then start it again to see what's
changed.");

        await _ec2Wrapper.StopInstances(instanceId);

        Console.WriteLine("Now let's start it up again.");
        await _ec2Wrapper.StartInstances(instanceId);

        Console.WriteLine("\nLet's see what changed.");

        instance = await _ec2Wrapper.DescribeInstance(instanceId);
        _uiMethods.DisplayTitle("New Instance Information");
        _ec2Wrapper.DisplayInstanceInformation(instance);

        Console.WriteLine("\nNotice the change in the SSH information:");
        Console.WriteLine(
            $"\\tssh -i {tempFileName} ec2-user@{instance.PublicIpAddress}");

        _uiMethods.PressEnter(isInteractive);

        Console.WriteLine(
            "Now we will stop the instance again. Then we will create and
associate an");
        Console.WriteLine("Elastic IP address to use with our instance.");

        await _ec2Wrapper.StopInstances(instanceId);
        _uiMethods.PressEnter(isInteractive);
```

```

        _uiMethods.DisplayTitle("Allocate Elastic IP address");
        Console.WriteLine(
            "You can allocate an Elastic IP address and associate it with your
instance\nto keep a consistent IP address even when your instance restarts.");
        var allocationResponse = await _ec2Wrapper.AllocateAddress();
        allocationId = allocationResponse.AllocationId;
        Console.WriteLine(
            "Now we will associate the Elastic IP address with our instance.");
        associationId = await _ec2Wrapper.AssociateAddress(allocationId,
instanceId);

        // Start the instance again.
        Console.WriteLine("Now let's start the instance again.");
        await _ec2Wrapper.StartInstances(instanceId);

        Console.WriteLine("\nLet's see what changed.");

        instance = await _ec2Wrapper.DescribeInstance(instanceId);
        _uiMethods.DisplayTitle("Instance information");
        _ec2Wrapper.DisplayInstanceInformation(instance);

        Console.WriteLine("\nHere is the SSH information:");
        Console.WriteLine(
            $"\"tssh -i {tempFileName} ec2-user@{instance.PublicIpAddress}\"");

        Console.WriteLine("Let's stop and start the instance again.");
        _uiMethods.PressEnter(isInteractive);

        await _ec2Wrapper.StopInstances(instanceId);

        Console.WriteLine("\nThe instance has stopped.");

        Console.WriteLine("Now let's start it up again.");
        await _ec2Wrapper.StartInstances(instanceId);

        instance = await _ec2Wrapper.DescribeInstance(instanceId);
        _uiMethods.DisplayTitle("New Instance Information");
        _ec2Wrapper.DisplayInstanceInformation(instance);
        Console.WriteLine("Note that the IP address did not change this time.");
        _uiMethods.PressEnter(isInteractive);

        await Cleanup();
    }
    catch (Exception ex)

```

```

        {
            _logger.LogError(ex, "There was a problem with the scenario, starting
cleanup.");
            await Cleanup();
        }

        _uiMethods.DisplayTitle("EC2 Basics Scenario completed.");
        _uiMethods.PressEnter(isInteractive);
    }

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host"></param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<EC2Basics>(loggerFactory);

        // Now the client is available for injection.
        _ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
        _ssmWrapper = host.Services.GetRequiredService<SsmWrapper>();
        _uiMethods = new UiMethods();
    }

    /// <summary>
    /// Clean up any resources from the scenario.
    /// </summary>
    /// <returns></returns>
    public static async Task Cleanup()
    {
        _uiMethods.DisplayTitle("Clean up resources");
        Console.WriteLine("Now let's clean up the resources we created.");

        Console.WriteLine("Disassociate the Elastic IP address and release it.");
        // Disassociate the Elastic IP address.
        await _ec2Wrapper.DisassociateIp(associationId);

        // Delete the Elastic IP address.
        await _ec2Wrapper.ReleaseAddress(allocationId);
    }

```

```

        // Terminate the instance.
        Console.WriteLine("Terminating the instance we created.");
        await _ec2Wrapper.TerminateInstances(instanceId);

        // Delete the security group.
        Console.WriteLine($"Deleting the Security Group: {groupName}.");
        await _ec2Wrapper.DeleteSecurityGroup(secGroupId);

        // Delete the RSA key pair.
        Console.WriteLine($"Deleting the key pair: {keyPairName}");
        await _ec2Wrapper.DeleteKeyPair(keyPairName);
        Console.WriteLine("Deleting the temporary file with the key information.");
        _ec2Wrapper.DeleteTempFile(tempFileName);
        _uiMethods.PressEnter(isInteractive);
    }
}

```

Définissez une classe qui englobe les EC2 actions.

```

/// <summary>
/// Methods of this class perform Amazon Elastic Compute Cloud (Amazon EC2).
/// </summary>
public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEC2;
    private readonly ILogger<EC2Wrapper> _logger;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonScheduler">The injected EC2 client.</param>
    /// <param name="logger">The injected logger.</param>
    public EC2Wrapper(IAmazonEC2 amazonService, ILogger<EC2Wrapper> logger)
    {
        _amazonEC2 = amazonService;
        _logger = logger;
    }

    /// <summary>
    /// Allocates an Elastic IP address that can be associated with an Amazon EC2
    /// instance. By using an Elastic IP address, you can keep the public IP address
    /// constant even when you restart the associated instance.

```

```

    /// </summary>
    /// <returns>The response object for the allocated address.</returns>
    public async Task<AllocateAddressResponse> AllocateAddress()
    {
        var request = new AllocateAddressRequest();

        try
        {
            var response = await _amazonEC2.AllocateAddressAsync(request);
            Console.WriteLine($"Allocated IP: {response.PublicIp} with allocation ID {response.AllocationId}.");
            return response;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "AddressLimitExceeded")
            {
                // For more information on Elastic IP address quotas, see:
                // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html#using-instance-addressing-limit
                _logger.LogError($"Unable to allocate Elastic IP, address limit exceeded. {ec2Exception.Message}");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while allocating Elastic IP.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Associates an Elastic IP address with an instance. When this association is
    /// created, the Elastic IP's public IP address is immediately used as the
    public
    /// IP address of the associated instance.
    /// </summary>
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
    param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>

```



```

    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
instanceId)
    {
        try
        {
            var request = new AssociateAddressRequest
            {
                AllocationId = allocationId,
                InstanceId = instanceId
            };

            var response = await _amazonEC2.AssociateAddressAsync(request);
            return response.AssociationId;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceId")
            {
                _logger.LogError(
                    $"InstanceId is invalid, unable to associate address.
{ec2Exception.Message}");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while associating the Elastic IP.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Authorize the local computer ingress to EC2 instances associated
    /// with the virtual private cloud (VPC) security group.
    /// </summary>
    /// <param name="groupName">The name of the security group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
    {

```

```

    try
    {
        // Get the IP address for the local computer.
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges =
            new List<IpRange> { new IpRange { CidrIp = $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidPermission.Duplicate")
        {
            _logger.LogError(
                $"The ingress rule already exists. {ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while authorizing ingress.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Authorize the local computer for ingress to
/// the Amazon EC2 SecurityGroup.
/// </summary>
/// <returns>The IPv4 address of the computer running the scenario.</returns>
private static async Task<string> GetIpAddress()
{

```

```

        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }

    /// <summary>
    /// Create an Amazon EC2 key pair with a specified name.
    /// </summary>
    /// <param name="keyPairName">The name for the new key pair.</param>
    /// <returns>The Amazon EC2 key pair created.</returns>
    public async Task<KeyPair?> CreateKeyPair(string keyPairName)
    {
        try
        {
            var request = new CreateKeyPairRequest { KeyName = keyPairName, };

            var response = await _amazonEC2.CreateKeyPairAsync(request);

            var kp = response.KeyPair;
            // Return the key pair so it can be saved if needed.

            // Wait until the key pair exists.
            int retries = 5;
            while (retries-- > 0)
            {
                Console.WriteLine($"Checking for new KeyPair {keyPairName}...");
                var keyPairs = await DescribeKeyPairs(keyPairName);
                if (keyPairs.Any())
                {
                    return kp;
                }

                Thread.Sleep(5000);
                retries--;
            }
            _logger.LogError($"Unable to find newly created KeyPair
{keyPairName}.");
            throw new DoesNotExistException("KeyPair not found");
        }
    }

```

```

        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidKeyPair.Duplicate")
            {
                _logger.LogError(
                    $"A key pair called {keyPairName} already exists.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while creating the key pair.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Save KeyPair information to a temporary file.
    /// </summary>
    /// <param name="keyPair">The name of the key pair.</param>
    /// <returns>The full path to the temporary file.</returns>
    public string SaveKeyPair(KeyPair keyPair)
    {
        var tempPath = Path.GetTempPath();
        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);

        return pemFileName;
    }

    /// <summary>
    /// Create an Amazon EC2 security group with a specified name and description.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>

```

```
public async Task<string> CreateSecurityGroup(string groupName, string
groupDescription)
{
    try
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        // Wait until the security group exists.
        int retries = 5;
        while (retries-- > 0)
        {
            var groups = await DescribeSecurityGroups(response.GroupId);
            if (groups.Any())
            {
                return response.GroupId;
            }

            Thread.Sleep(5000);
            retries--;
        }
        _logger.LogError($"Unable to find newly created group {groupName}.");
        throw new DoesNotExistException("security group not found");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "ResourceAlreadyExists")
        {
            _logger.LogError(
                $"A security group with the name {groupName} already exists.
{ec2Exception.Message}");
        }
        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while creating the security group.:
{ex.Message}");
        throw;
    }
}
```

```

/// <summary>
/// Create a new Amazon EC2 VPC.
/// </summary>
/// <param name="cidrBlock">The CIDR block for the new security group.</param>
/// <returns>The VPC Id of the new VPC.</returns>
public async Task<string?> CreateVPC(string cidrBlock)
{
    try
    {
        var response = await _amazonEC2.CreateVpcAsync(new CreateVpcRequest
        {
            CidrBlock = cidrBlock,
        });

        Vpc vpc = response.Vpc;
        Console.WriteLine($"Created VPC with ID: {vpc.VpcId}.");
        return vpc.VpcId;
    }
    catch (AmazonEC2Exception ex)
    {
        Console.WriteLine($"Couldn't create VPC because: {ex.Message}");
        return null;
    }
}

/// <summary>
/// Delete an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyPair(string keyPairName)
{
    try
    {
        await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
        return true;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidKeyPair.NotFound")
        {

```

```

        _logger.LogError($"KeyPair {keyPairName} does not exist and cannot
be deleted. Please verify the key pair name and try again.");
    }

    return false;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
    return false;
}
}

/// <summary>
/// Delete the temporary file where the key pair information was saved.
/// </summary>
/// <param name="tempFileName">The path to the temporary file.</param>
public void DeleteTempFile(string tempFileName)
{
    if (File.Exists(tempFileName))
    {
        File.Delete(tempFileName);
    }
}

/// <summary>
/// Delete an Amazon EC2 security group.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteSecurityGroup(string groupId)
{
    try
    {
        var response =
            await _amazonEC2.DeleteSecurityGroupAsync(
                new DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidGroup.NotFound")
        {

```

```

        _logger.LogError(
            $"Security Group {groupId} does not exist and cannot be deleted.
Please verify the ID and try again.");
    }

    return false;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't delete the security group because:
{ex.Message}");
    return false;
}
}

/// <summary>
/// Delete an Amazon EC2 VPC.
/// </summary>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteVpc(string vpcId)
{
    var request = new DeleteVpcRequest
    {
        VpcId = vpcId,
    };

    var response = await _amazonEC2.DeleteVpcAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Get information about existing Amazon EC2 images.
/// </summary>
/// <returns>A list of image information.</returns>
public async Task<List<Image>> DescribeImages(List<string>? imageIds)
{
    var request = new DescribeImagesRequest();
    if (imageIds is not null)
    {
        // If the imageIds list is not null, add the list
        // to the request object.
        request.ImageIds = imageIds;
    }
}

```



```

        var response = await _amazonEC2.DescribeImagesAsync(request);
        return response.Images;
    }

    /// <summary>
    /// Display the information returned by DescribeImages.
    /// </summary>
    /// <param name="images">The list of image information to display.</param>
    public void DisplayImageInfo(List<Image> images)
    {
        images.ForEach(image =>
        {
            Console.WriteLine($"{image.Name} Created on: {image.CreationDate}");
        });
    }

    /// <summary>
    /// Get information about an Amazon EC2 instance.
    /// </summary>
    /// <param name="instanceId">The instance Id of the EC2 instance.</param>
    /// <returns>An EC2 instance.</returns>
    public async Task<Instance> DescribeInstance(string instanceId)
    {
        var response = await _amazonEC2.DescribeInstancesAsync(
            new DescribeInstancesRequest { InstanceIds = new List<string>
{ instanceId } });
        return response.Reservations[0].Instances[0];
    }

    /// <summary>
    /// Display EC2 instance information.
    /// </summary>
    /// <param name="instance">The instance Id of the EC2 instance.</param>
    public void DisplayInstanceInformation(Instance instance)
    {
        Console.WriteLine($"ID: {instance.InstanceId}");
        Console.WriteLine($"Image ID: {instance.ImageId}");
        Console.WriteLine($"{instance.InstanceType}");
        Console.WriteLine($"Key Name: {instance.KeyName}");
        Console.WriteLine($"VPC ID: {instance.VpcId}");
        Console.WriteLine($"Public IP: {instance.PublicIpAddress}");
        Console.WriteLine($"State: {instance.State.Name}");
    }

```

```

    }

    /// <summary>
    /// Get information about EC2 instances with a particular state.
    /// </summary>
    /// <param name="tagName">The name of the tag to filter on.</param>
    /// <param name="tagValue">The value of the tag to look for.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> GetInstancesWithState(string state)
    {
        try
        {
            // Filters the results of the instance list.
            var filters = new List<Filter>
            {
                new Filter
                {
                    Name = $"instance-state-name",
                    Values = new List<string> { state, },
                },
            };
            var request = new DescribeInstancesRequest { Filters = filters, };

            Console.WriteLine($"\\nShowing instances with state {state}");
            var paginator = _amazonEC2.Paginators.DescribeInstances(request);

            await foreach (var response in paginator.Responses)
            {
                foreach (var reservation in response.Reservations)
                {
                    foreach (var instance in reservation.Instances)
                    {
                        Console.Write($"Instance ID: {instance.InstanceId} ");
                        Console.WriteLine($"\\tCurrent State:
{instance.State.Name}");
                    }
                }
            }

            return true;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidParameterValue")

```

```

        {
            _logger.LogError(
                $"Invalid parameter value for filtering instances.");
        }

        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't list instances because: {ex.Message}");
        return false;
    }
}

/// <summary>
/// Describe the instance types available.
/// </summary>
/// <returns>A list of instance type information.</returns>
public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
{
    try
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
        {
            new Filter("processor-info.supported-architecture",
                new List<string> { architecture.ToString() })
        };
        filters.Add(new Filter("instance-type", new() { "*.micro",
            "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();

        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
        {
            instanceTypes.Add(instanceType);
        }

        return instanceTypes;
    }
}

```

```

        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidParameterValue")
            {
                _logger.LogError(
                    $"Parameters are invalid. Ensure architecture and size strings
conform to DescribeInstanceTypes API reference.");
            }

            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete the security group because:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get information about an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair.</param>
    /// <returns>A list of key pair information.</returns>
    public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
    {
        try
        {
            var request = new DescribeKeyPairsRequest();
            if (!string.IsNullOrEmpty(keyPairName))
            {
                request = new DescribeKeyPairsRequest
                {
                    KeyNames = new List<string> { keyPairName }
                };
            }

            var response = await _amazonEC2.DescribeKeyPairsAsync(request);
            return response.KeyPairs.ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidKeyPair.NotFound")
            {

```

```

        _logger.LogError(
            $"A key pair called {keyPairName} does not exist.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while describing the key pair.: {ex.Message}");
    throw;
}
}

/// <summary>
/// Retrieve information for one or all Amazon EC2 security group.
/// </summary>
/// <param name="groupId">The optional Id of a specific Amazon EC2 security
group.</param>
/// <returns>A list of security group information.</returns>
public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
{
    try
    {
        var securityGroups = new List<SecurityGroup>();
        var request = new DescribeSecurityGroupsRequest();

        if (!string.IsNullOrEmpty(groupId))
        {
            var groupIds = new List<string> { groupId };
            request.GroupIds = groupIds;
        }

        var paginatorForSecurityGroups =
            _amazonEC2.Paginators.DescribeSecurityGroups(request);

        await foreach (var securityGroup in
paginatorForSecurityGroups.SecurityGroups)
        {
            securityGroups.Add(securityGroup);
        }

        return securityGroups;
    }
}

```

```

    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidGroup.NotFound")
        {
            _logger.LogError(
                $"A security group {groupId} does not exist.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while listing security groups. {ex.Message}");
        throw;
    }
}

/// <summary>
/// Display the information returned by the call to
/// DescribeSecurityGroupsAsync.
/// </summary>
/// <param name="securityGroup">A list of security group information.</param>
public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
{
    Console.WriteLine($"{securityGroup.GroupName}");
    Console.WriteLine("Ingress permissions:");
    securityGroup.IpPermissions.ForEach(permission =>
    {
        Console.WriteLine($"    \tFromPort: {permission.FromPort}");
        Console.WriteLine($"    \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"    \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"    \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.Write($"{range.CidrIpv6} "); });

        Console.WriteLine($"    \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));
    });
}

```

```

        Console.WriteLine($"\\n\\tTo Port: {permission.ToPort}");
    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"\\tFromPort: {permission.FromPort}");
        Console.WriteLine($"\\tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"\\tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"\\n\\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.Write($"{range.CidrIpv6} "); });

        Console.WriteLine($"\\n\\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

        Console.WriteLine($"\\n\\tTo Port: {permission.ToPort}");
    });
}

/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    try
    {
        var response = await _amazonEC2.DisassociateAddressAsync(
            new DisassociateAddressRequest { AssociationId = associationId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidAssociationID.NotFound")
        {
            _logger.LogError(

```

```

        $"AssociationId is invalid, unable to disassociate address.
{ec2Exception.Message}");
    }

    return false;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while disassociating the Elastic IP.:
{ex.Message}");
    return false;
}
}

/// <summary>
/// Reboot a specific EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the instance that will be
rebooted.</param>
/// <returns>Async Task.</returns>
public async Task<bool> RebootInstances(string ec2InstanceId)
{
    try
    {
        var request = new RebootInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        await _amazonEC2.RebootInstancesAsync(request);

        // Wait for the instance to be running.
        Console.WriteLine("Waiting for the instance to start.");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Running);

        return true;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(

```



```

        $"InstanceId {ec2InstanceId} is invalid, unable to reboot.
{ec2Exception.Message}");
    }
    return false;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while rebooting the instance {ec2InstanceId}.:
{ex.Message}");
    return false;
}
}

/// <summary>
/// Release an Elastic IP address. After the Elastic IP address is released,
/// it can no longer be used.
/// </summary>
/// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> ReleaseAddress(string allocationId)
{
    try
    {
        var request = new ReleaseAddressRequest { AllocationId = allocationId };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidAllocationID.NotFound")
        {
            _logger.LogError(
                $"AllocationId {allocationId} was not found.
{ec2Exception.Message}");
        }

        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError(

```

```

        $"An error occurred while releasing the AllocationId
{allocationId}.: {ex.Message}");
        return false;
    }
}

/// <summary>
/// Create and run an EC2 instance.
/// </summary>
/// <param name="ImageId">The image Id of the image used as a basis for the
/// EC2 instance.</param>
/// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
/// <param name="keyName">The name of the key pair to associate with the
/// instance.</param>
/// <param name="groupId">The Id of the Amazon EC2 security group that will be
/// allowed to interact with the new EC2 instance.</param>
/// <returns>The instance Id of the new EC2 instance.</returns>
public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
{
    try
    {
        var request = new RunInstancesRequest
        {
            ImageId = imageId,
            InstanceType = instanceType,
            KeyName = keyName,
            MinCount = 1,
            MaxCount = 1,
            SecurityGroupIds = new List<string> { groupId }
        };
        var response = await _amazonEC2.RunInstancesAsync(request);
        var instanceId = response.Reservation.Instances[0].InstanceId;

        Console.WriteLine("Waiting for the instance to start.");
        await WaitForInstanceState(instanceId, InstanceStateName.Running);

        return instanceId;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidGroupId.NotFound")
        {

```

```

        _logger.LogError(
            $"GroupId {groupId} was not found. {ec2Exception.Message}");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while running the instance.: {ex.Message}");
    throw;
}
}

/// <summary>
/// Start an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
/// to start.</param>
/// <returns>Async task.</returns>
public async Task StartInstances(string ec2InstanceId)
{
    try
    {
        var request = new StartInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        await _amazonEC2.StartInstancesAsync(request);

        Console.WriteLine("Waiting for instance to start. ");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Running);
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(
                $"InstanceId is invalid, unable to start.
{ec2Exception.Message}");
        }
    }
}

```

```

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while starting the instance.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Stop an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance to
/// stop.</param>
/// <returns>Async task.</returns>
public async Task StopInstances(string ec2InstanceId)
{
    try
    {
        var request = new StopInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        await _amazonEC2.StopInstancesAsync(request);
        Console.Write("Waiting for the instance to stop.");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Stopped);

        Console.WriteLine("\nThe instance has stopped.");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(
                $"InstanceId is invalid, unable to stop.
{ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {

```

```

        _logger.LogError(
            $"An error occurred while stopping the instance.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Terminate an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance
/// to terminate.</param>
/// <returns>Async task.</returns>
public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
{
    try
    {
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };

        var response = await _amazonEC2.TerminateInstancesAsync(request);
        Console.Write("Waiting for the instance to terminate.");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Terminated);

        Console.WriteLine($"\\nThe instance {ec2InstanceId} has been
terminated.");
        return response.TerminatingInstances;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(
                $"InstanceId is invalid, unable to terminate.
{ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(

```

```

        $"An error occurred while terminating the instance.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{
    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;
    do
    {
        // Wait 5 seconds.
        Thread.Sleep(5000);

        // Check for the desired state.
        var response = await _amazonEC2.DescribeInstancesAsync(request);
        var instance = response.Reservations[0].Instances[0];
        hasState = instance.State.Name == stateName;
        Console.Write(". ");
    } while (!hasState);

    return hasState;
}
}

```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [AllocateAddress](#)

- [AssociateAddress](#)
- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Actions

AllocateAddress

L'exemple de code suivant montre comment utiliser `AllocateAddress`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
```

```

/// Allocates an Elastic IP address that can be associated with an Amazon EC2
/// instance. By using an Elastic IP address, you can keep the public IP address
/// constant even when you restart the associated instance.
/// </summary>
/// <returns>The response object for the allocated address.</returns>
public async Task<AllocateAddressResponse> AllocateAddress()
{
    var request = new AllocateAddressRequest();

    try
    {
        var response = await _amazonEC2.AllocateAddressAsync(request);
        Console.WriteLine($"Allocated IP: {response.PublicIp} with allocation ID
{response.AllocationId}.");
        return response;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "AddressLimitExceeded")
        {
            // For more information on Elastic IP address quotas, see:
            // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-
addresses-eip.html#using-instance-addressing-limit
            _logger.LogError($"Unable to allocate Elastic IP, address limit
exceeded. {ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while allocating Elastic IP.:
{ex.Message}");
        throw;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [AllocateAddress](#) à la section Référence des AWS SDK pour .NET API.

AssociateAddress

L'exemple de code suivant montre comment utiliser `AssociateAddress`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Associates an Elastic IP address with an instance. When this association is
    /// created, the Elastic IP's public IP address is immediately used as the
    public
    /// IP address of the associated instance.
    /// </summary>
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
    param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>
    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
    instanceId)
    {
        try
        {
            var request = new AssociateAddressRequest
            {
                AllocationId = allocationId,
                InstanceId = instanceId
            };

            var response = await _amazonEC2.AssociateAddressAsync(request);
            return response.AssociationId;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceId")
            {
                _logger.LogError(

```

```

        $"InstanceId is invalid, unable to associate address.
{ec2Exception.Message}");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while associating the Elastic IP.:
{ex.Message}");
    throw;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [AssociateAddress](#) à la section Référence des AWS SDK pour .NET API.

AuthorizeSecurityGroupIngress

L'exemple de code suivant montre comment utiliser `AuthorizeSecurityGroupIngress`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Authorize the local computer ingress to EC2 instances associated
/// with the virtual private cloud (VPC) security group.
/// </summary>
/// <param name="groupName">The name of the security group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
{
    try
    {

```

```

        // Get the IP address for the local computer.
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges =
            new List<IpRange> { new IpRange { CidrIp = $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidPermission.Duplicate")
        {
            _logger.LogError(
                $"The ingress rule already exists. {ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while authorizing ingress.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Authorize the local computer for ingress to
/// the Amazon EC2 SecurityGroup.
/// </summary>
/// <returns>The IPv4 address of the computer running the scenario.</returns>
private static async Task<string> GetIpAddress()
{
    var httpClient = new HttpClient();

```

```

        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }

```

- Pour plus de détails sur l'API, reportez-vous [AuthorizeSecurityGroupIngress](#) à la section Référence des AWS SDK pour .NET API.

CreateKeyPair

L'exemple de code suivant montre comment utiliser `CreateKeyPair`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create an Amazon EC2 key pair with a specified name.
/// </summary>
/// <param name="keyPairName">The name for the new key pair.</param>
/// <returns>The Amazon EC2 key pair created.</returns>
public async Task<KeyPair?> CreateKeyPair(string keyPairName)
{
    try
    {
        var request = new CreateKeyPairRequest { KeyName = keyPairName, };

        var response = await _amazonEC2.CreateKeyPairAsync(request);

        var kp = response.KeyPair;
        // Return the key pair so it can be saved if needed.
    }
}

```

```

        // Wait until the key pair exists.
        int retries = 5;
        while (retries-- > 0)
        {
            Console.WriteLine($"Checking for new KeyPair {keyPairName}...");
            var keyPairs = await DescribeKeyPairs(keyPairName);
            if (keyPairs.Any())
            {
                return kp;
            }

            Thread.Sleep(5000);
            retries--;
        }
        _logger.LogError($"Unable to find newly created KeyPair
{keyPairName}.");
        throw new DoesNotExistException("KeyPair not found");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidKeyPair.Duplicate")
        {
            _logger.LogError(
                $"A key pair called {keyPairName} already exists.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while creating the key pair.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Save KeyPair information to a temporary file.
/// </summary>
/// <param name="keyPair">The name of the key pair.</param>
/// <returns>The full path to the temporary file.</returns>
public string SaveKeyPair(KeyPair keyPair)
{
    var tempPath = Path.GetTempPath();

```

```

        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);

        return pemFileName;
    }

```

- Pour plus de détails sur l'API, reportez-vous [CreateKeyPair](#) à la section Référence des AWS SDK pour .NET API.

CreateLaunchTemplate

L'exemple de code suivant montre comment utiliser `CreateLaunchTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
    after
    /// the instance is started. This script installs the Python packages and starts
    a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
    run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
    and attach to the profile.</param>
    /// <returns>The template object.</returns>

```

```

    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);
            await CreateInstanceProfileWithName(_instancePolicyName,
_instanceRoleName,
            _instanceProfileName, instancePolicyPath);

            var startServerText = await File.ReadAllTextAsync(startupScriptPath);
            var plainTextBytes =
System.Text.Encoding.UTF8.GetBytes(startServerText);

            var amiLatest = await _amazonSsm.GetParameterAsync(
                new GetParameterRequest() { Name = _amiParam });
            var amiId = amiLatest.Parameter.Value;
            var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
                new CreateLaunchTemplateRequest()
                {
                    LaunchTemplateName = _launchTemplateName,
                    LaunchTemplateData = new RequestLaunchTemplateData()
                    {
                        InstanceType = _instanceType,
                        ImageId = amiId,
                        IamInstanceProfile =
                            new
LaunchTemplateIamInstanceProfileSpecificationRequest()
                            {
                                Name = _instanceProfileName
                            },
                        KeyName = _keyPairName,
                        UserData = System.Convert.ToBase64String(plainTextBytes)
                    }
                });
            return launchTemplateResponse.LaunchTemplate;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.AlreadyExistsException")
            {

```

```

        _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                        $"Please try again with a unique name.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
    throw;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateLaunchTemplate](#) à la section Référence des AWS SDK pour .NET API.

CreateSecurityGroup

L'exemple de code suivant montre comment utiliser `CreateSecurityGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create an Amazon EC2 security group with a specified name and description.
/// </summary>
/// <param name="groupName">The name for the new security group.</param>
/// <param name="groupDescription">A description of the new security group.</
param>
/// <returns>The group Id of the new security group.</returns>
public async Task<string> CreateSecurityGroup(string groupName, string
groupDescription)
{

```



```
try
{
    var response = await _amazonEC2.CreateSecurityGroupAsync(
        new CreateSecurityGroupRequest(groupName, groupDescription));

    // Wait until the security group exists.
    int retries = 5;
    while (retries-- > 0)
    {
        var groups = await DescribeSecurityGroups(response.GroupId);
        if (groups.Any())
        {
            return response.GroupId;
        }

        Thread.Sleep(5000);
        retries--;
    }
    _logger.LogError($"Unable to find newly created group {groupName}.");
    throw new DoesNotExistException("security group not found");
}
catch (AmazonEC2Exception ec2Exception)
{
    if (ec2Exception.ErrorCode == "ResourceAlreadyExists")
    {
        _logger.LogError(
            $"A security group with the name {groupName} already exists.
{ec2Exception.Message}");
    }
    throw;
}
catch (Exception ex)
{
    _logger.LogError(
        $"An error occurred while creating the security group.:
{ex.Message}");
    throw;
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateSecurityGroup](#) à la section Référence des AWS SDK pour .NET API.

DeleteKeyPair

L'exemple de code suivant montre comment utiliser `DeleteKeyPair`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyPair(string keyPairName)
{
    try
    {
        await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
        return true;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidKeyPair.NotFound")
        {
            _logger.LogError($"KeyPair {keyPairName} does not exist and cannot
be deleted. Please verify the key pair name and try again.");
        }

        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
    }
}
```

```

        return false;
    }
}

/// <summary>
/// Delete the temporary file where the key pair information was saved.
/// </summary>
/// <param name="tempFileName">The path to the temporary file.</param>
public void DeleteTempFile(string tempFileName)
{
    if (File.Exists(tempFileName))
    {
        File.Delete(tempFileName);
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteKeyPair](#) à la section Référence des AWS SDK pour .NET API.

DeleteLaunchTemplate

L'exemple de code suivant montre comment utiliser `DeleteLaunchTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {

```

```
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
            "InvalidLaunchTemplateName.NotFoundException")
        {
            _logger.LogError(
                $"Could not delete the template, the name {_launchTemplateName}
was not found.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
        throw;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteLaunchTemplate](#) à la section Référence des AWS SDK pour .NET API.

DeleteSecurityGroup

L'exemple de code suivant montre comment utiliser `DeleteSecurityGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Delete an Amazon EC2 security group.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteSecurityGroup(string groupId)
{
    try
    {
        var response =
            await _amazonEC2.DeleteSecurityGroupAsync(
                new DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidGroup.NotFound")
        {
            _logger.LogError(
                $"Security Group {groupId} does not exist and cannot be deleted.
Please verify the ID and try again.");
        }

        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete the security group because:
{ex.Message}");
        return false;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteSecurityGroup](#) à la section Référence des AWS SDK pour .NET API.

DescribeAvailabilityZones

L'exemple de code suivant montre comment utiliser `DescribeAvailabilityZones`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    try
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while listing availability zones.:
{ex.Message}");
        throw;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeAvailabilityZones](#) à la section Référence des AWS SDK pour .NET API.

DescribeIamInstanceProfileAssociations

L'exemple de code suivant montre comment utiliser `DescribeIamInstanceProfileAssociations`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>
/// <returns>Instance profile associations data.</returns>
public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
{
    try
    {
        var response = await
        _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
}
```

```

    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while creating the template.:
{ex.Message}");
        throw;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeInstanceProfileAssociations](#) à la section Référence des AWS SDK pour .NET API.

DescribeInstanceTypes

L'exemple de code suivant montre comment utiliser `DescribeInstanceTypes`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Describe the instance types available.
/// </summary>
/// <returns>A list of instance type information.</returns>
public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
{
    try
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
        {
            new Filter("processor-info.supported-architecture",
                new List<string> { architecture.ToString() })
        };
    }
}

```



```

        filters.Add(new Filter("instance-type", new() { "*.micro",
        "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();

        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
        {
            instanceTypes.Add(instanceType);
        }

        return instanceTypes;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidParameterValue")
        {
            _logger.LogError(
                $"Parameters are invalid. Ensure architecture and size strings
conform to DescribeInstanceTypes API reference.");
        }

        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete the security group because:
{ex.Message}");
        throw;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeInstanceTypes](#) à la section Référence des AWS SDK pour .NET API.

DescribeInstances

L'exemple de code suivant montre comment utiliser `DescribeInstances`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get information about EC2 instances with a particular state.
/// </summary>
/// <param name="tagName">The name of the tag to filter on.</param>
/// <param name="tagValue">The value of the tag to look for.</param>
/// <returns>True if successful.</returns>
public async Task<bool> GetInstancesWithState(string state)
{
    try
    {
        // Filters the results of the instance list.
        var filters = new List<Filter>
        {
            new Filter
            {
                Name = $"instance-state-name",
                Values = new List<string> { state, },
            },
        };
        var request = new DescribeInstancesRequest { Filters = filters, };

        Console.WriteLine($"\\nShowing instances with state {state}");
        var paginator = _amazonEC2.Paginators.DescribeInstances(request);

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\\tCurrent State:
{instance.State.Name}");
                }
            }
        }
    }
}

```

```
        }
    }

    return true;
}
catch (AmazonEC2Exception ec2Exception)
{
    if (ec2Exception.ErrorCode == "InvalidParameterValue")
    {
        _logger.LogError(
            $"Invalid parameter value for filtering instances.");
    }

    return false;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't list instances because: {ex.Message}");
    return false;
}
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeInstances](#) à la section Référence des AWS SDK pour .NET API.

DescribeKeyPairs

L'exemple de code suivant montre comment utiliser `DescribeKeyPairs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information about an Amazon EC2 key pair.
/// </summary>
```

```
/// <param name="keyPairName">The name of the key pair.</param>
/// <returns>A list of key pair information.</returns>
public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
{
    try
    {
        var request = new DescribeKeyPairsRequest();
        if (!string.IsNullOrEmpty(keyPairName))
        {
            request = new DescribeKeyPairsRequest
            {
                KeyNames = new List<string> { keyPairName }
            };
        }

        var response = await _amazonEC2.DescribeKeyPairsAsync(request);
        return response.KeyPairs.ToList();
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidKeyPair.NotFound")
        {
            _logger.LogError(
                $"A key pair called {keyPairName} does not exist.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while describing the key pair.: {ex.Message}");
        throw;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeKeyPairs](#) à la section Référence des AWS SDK pour .NET API.

DescribeSecurityGroups

L'exemple de code suivant montre comment utiliser `DescribeSecurityGroups`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Retrieve information for one or all Amazon EC2 security group.
/// </summary>
/// <param name="groupId">The optional Id of a specific Amazon EC2 security
group.</param>
/// <returns>A list of security group information.</returns>
public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
{
    try
    {
        var securityGroups = new List<SecurityGroup>();
        var request = new DescribeSecurityGroupsRequest();

        if (!string.IsNullOrEmpty(groupId))
        {
            var groupIds = new List<string> { groupId };
            request.GroupIds = groupIds;
        }

        var paginatorForSecurityGroups =
            _amazonEC2.Paginators.DescribeSecurityGroups(request);

        await foreach (var securityGroup in
            paginatorForSecurityGroups.SecurityGroups)
        {
            securityGroups.Add(securityGroup);
        }

        return securityGroups;
    }
}
```

```

        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidGroup.NotFound")
            {
                _logger.LogError(
                    $"A security group {groupId} does not exist.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while listing security groups. {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Display the information returned by the call to
    /// DescribeSecurityGroupsAsync.
    /// </summary>
    /// <param name="securityGroup">A list of security group information.</param>
    public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
    {
        Console.WriteLine($"{securityGroup.GroupName}");
        Console.WriteLine("Ingress permissions:");
        securityGroup.IpPermissions.ForEach(permission =>
        {
            Console.WriteLine($"  \tFromPort: {permission.FromPort}");
            Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

            Console.WriteLine($"  \tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}

"); });

            Console.WriteLine($"  \n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
            { Console.Write($"{range.CidrIpv6} "); });

            Console.WriteLine($"  \n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");

```

```

    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"    \tFromPort: {permission.FromPort}");
        Console.WriteLine($"    \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"    \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"    \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
        { Console.Write($"{range.CidrIpv6} "); });

        Console.WriteLine($"    \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

        Console.WriteLine($"    \n\tTo Port: {permission.ToPort}");
    });
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeSecurityGroups](#) à la section Référence des AWS SDK pour .NET API.

DescribeSubnets

L'exemple de code suivant montre comment utiliser `DescribeSubnets`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.

```

```

    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        try
        {
            var subnets = new List<Subnet>();
            var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
                new DescribeSubnetsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("vpc-id", new List<string>() { vpcId }),
                        new("availability-zone", availabilityZones),
                        new("default-for-az", new List<string>() { "true" })
                    }
                });

            // Get the entire list using the paginator.
            await foreach (var subnet in subnetPaginator.Subnets)
            {
                subnets.Add(subnet);
            }

            return subnets;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
            {
                _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId} does
not exist.");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while describing the subnets.:
{ex.Message}");
            throw;
        }
    }

```



```
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeSubnets](#) à la section Référence des AWS SDK pour .NET API.

DescribeVpcs

L'exemple de code suivant montre comment utiliser `DescribeVpcs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Get the default VPC for the account.  
/// </summary>  
/// <returns>The default VPC object.</returns>  
public async Task<Vpc> GetDefaultVpc()  
{  
    try  
    {  
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(  
            new DescribeVpcsRequest()  
            {  
                Filters = new List<Amazon.EC2.Model.Filter>()  
                {  
                    new("is-default", new List<string>() { "true" })  
                }  
            });  
        return vpcResponse.Vpcs[0];  
    }  
    catch (AmazonEC2Exception ec2Exception)  
    {  
        if (ec2Exception.ErrorCode == "UnauthorizedOperation")  
        {  

```

```

        _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
    throw;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeVpcs](#) à la section Référence des AWS SDK pour .NET API.

DisassociateAddress

L'exemple de code suivant montre comment utiliser `DisassociateAddress`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    try
    {
        var response = await _amazonEC2.DisassociateAddressAsync(
            new DisassociateAddressRequest { AssociationId = associationId });
    }
}

```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidAssociationID.NotFound")
        {
            _logger.LogError(
                $"AssociationId is invalid, unable to disassociate address.
{ec2Exception.Message}");
        }

        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while disassociating the Elastic IP.:
{ex.Message}");
        return false;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DisassociateAddress](#) à la section Référence des AWS SDK pour .NET API.

RebootInstances

L'exemple de code suivant montre comment utiliser `RebootInstances`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Redémarrez une instance à l'aide de son identifiant.

```
/// <summary>
/// Reboot a specific EC2 instance.
```

```

    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the instance that will be
rebooted.</param>
    /// <returns>Async Task.</returns>
    public async Task<bool> RebootInstances(string ec2InstanceId)
    {
        try
        {
            var request = new RebootInstancesRequest
            {
                InstanceIds = new List<string> { ec2InstanceId },
            };

            await _amazonEC2.RebootInstancesAsync(request);

            // Wait for the instance to be running.
            Console.WriteLine("Waiting for the instance to start.");
            await WaitForInstanceState(ec2InstanceId, InstanceStateName.Running);

            return true;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceId")
            {
                _logger.LogError(
                    $"InstanceId {ec2InstanceId} is invalid, unable to reboot.
{ec2Exception.Message}");
            }
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while rebooting the instance {ec2InstanceId}.:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>

```

```

    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
    InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is in the specified state.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEC2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.WriteLine(". ");
        } while (!hasState);

        return hasState;
    }

```

Remplacez le profil par une instance, redémarrez et redémarrez un serveur web.

```

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
    /// replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
    /// ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
    the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
    the instance.</param>

```

```

    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
instancesPaginator.InstanceInformationList)
                {
                    instanceReady = instance.InstanceId == instanceId;
                    if (instanceReady)
                    {
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");
            Console.WriteLine($"Sending restart command to instance {instanceId}");
            await _amazonSsm.SendCommandAsync(

```

```

        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
            DocumentName = "AWS-RunShellScript",
            Parameters = new Dictionary<string, List<string>>()
            {
                {
                    "commands",
                    new List<string>() { "cd / && sudo python3 server.py
80" }
                }
            }
        });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while replacing the template.:
{ex.Message}");
        throw;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [RebootInstances](#) à la section Référence des AWS SDK pour .NET API.

ReleaseAddress

L'exemple de code suivant montre comment utiliser `ReleaseAddress`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Release an Elastic IP address. After the Elastic IP address is released,
/// it can no longer be used.
/// </summary>
/// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> ReleaseAddress(string allocationId)
{
    try
    {
        var request = new ReleaseAddressRequest { AllocationId = allocationId };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidAllocationID.NotFound")
        {
            _logger.LogError(
                $"AllocationId {allocationId} was not found.
{ec2Exception.Message}");
        }

        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while releasing the AllocationId
{allocationId}.: {ex.Message}");
        return false;
    }
}
```



```
}
```

- Pour plus de détails sur l'API, reportez-vous [ReleaseAddress](#) à la section Référence des AWS SDK pour .NET API.

ReplaceIamInstanceProfileAssociation

L'exemple de code suivant montre comment utiliser `ReplaceIamInstanceProfileAssociation`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Replace the profile associated with a running instance. After the profile is
replaced, the instance
/// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
/// used to restart the Python web server.
/// </summary>
/// <param name="instanceId">The Id of the instance to update.</param>
/// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
/// <param name="associationId">The Id of the existing profile association for
the instance.</param>
/// <returns>Async task.</returns>
public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
{
    try
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
            new ReplaceIamInstanceProfileAssociationRequest()
            {
                AssociationId = associationId,
                IamInstanceProfile = new IamInstanceProfileSpecification()
```

```

        {
            Name = credsProfileName
        }
    });
    // Allow time before resetting.
    Thread.Sleep(25000);

    await _amazonEc2.RebootInstancesAsync(
        new RebootInstancesRequest(new List<string>() { instanceId }));
    Thread.Sleep(25000);
    var instanceReady = false;
    var retries = 5;
    while (retries-- > 0 && !instanceReady)
    {
        var instancesPaginator =
            _amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
        // Get the entire list using the paginator.
        await foreach (var instance in
instancesPaginator.InstanceInformationList)
        {
            instanceReady = instance.InstanceId == instanceId;
            if (instanceReady)
            {
                break;
            }
        }
    }
    Console.WriteLine("Waiting for instance to be running.");
    await WaitForInstanceState(instanceId, InstanceStateName.Running);
    Console.WriteLine("Instance ready.");
    Console.WriteLine($"Sending restart command to instance {instanceId}");
    await _amazonSsm.SendCommandAsync(
        new SendCommandRequest()
        {
            InstanceIds = new List<string>() { instanceId },
            DocumentName = "AWS-RunShellScript",
            Parameters = new Dictionary<string, List<string>>()
            {
                {
                    "commands",
                    new List<string>() { "cd / && sudo python3 server.py
80" }
                }
            }
        }
    );

```

```

        }
    });
    Console.WriteLine($"Restarted the web server on instance {instanceId}");
}
catch (AmazonEC2Exception ec2Exception)
{
    if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
    {
        _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while replacing the template.: {ex.Message}");
    throw;
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ReplacelamInstanceProfileAssociation](#) à la section Référence des AWS SDK pour .NET API.

RunInstances

L'exemple de code suivant montre comment utiliser `RunInstances`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create and run an EC2 instance.
/// </summary>
/// <param name="ImageId">The image Id of the image used as a basis for the

```

```

    /// EC2 instance.</param>
    /// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
    /// <param name="keyName">The name of the key pair to associate with the
    /// instance.</param>
    /// <param name="groupId">The Id of the Amazon EC2 security group that will be
    /// allowed to interact with the new EC2 instance.</param>
    /// <returns>The instance Id of the new EC2 instance.</returns>
    public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
    {
        try
        {
            var request = new RunInstancesRequest
            {
                ImageId = imageId,
                InstanceType = instanceType,
                KeyName = keyName,
                MinCount = 1,
                MaxCount = 1,
                SecurityGroupIds = new List<string> { groupId }
            };
            var response = await _amazonEC2.RunInstancesAsync(request);
            var instanceId = response.Reservation.Instances[0].InstanceId;

            Console.WriteLine("Waiting for the instance to start.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);

            return instanceId;
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidGroupId.NotFound")
            {
                _logger.LogError(
                    $"GroupId {groupId} was not found. {ec2Exception.Message}");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while running the instance.: {ex.Message}");
        }
    }

```

```
        throw;  
    }  
}
```

- Pour plus de détails sur l'API, reportez-vous [RunInstances](#) à la section Référence des AWS SDK pour .NET API.

StartInstances

L'exemple de code suivant montre comment utiliser `StartInstances`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Start an EC2 instance.  
/// </summary>  
/// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance  
/// to start.</param>  
/// <returns>Async task.</returns>  
public async Task StartInstances(string ec2InstanceId)  
{  
    try  
    {  
        var request = new StartInstancesRequest  
        {  
            InstanceIds = new List<string> { ec2InstanceId },  
        };  
  
        await _amazonEC2.StartInstancesAsync(request);  
  
        Console.WriteLine("Waiting for instance to start. ");  
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Running);  
    }  
}
```

```

        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceId")
            {
                _logger.LogError(
                    $"InstanceId is invalid, unable to start.
{ec2Exception.Message}");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while starting the instance.: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
    InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is in the specified state.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEC2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
        }
    }

```

```

        Console.Write(". ");
    } while (!hasState);

    return hasState;
}

```

- Pour plus de détails sur l'API, reportez-vous [StartInstances](#) à la section Référence des AWS SDK pour .NET API.

StopInstances

L'exemple de code suivant montre comment utiliser `StopInstances`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Stop an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance to
/// stop.</param>
/// <returns>Async task.</returns>
public async Task StopInstances(string ec2InstanceId)
{
    try
    {
        var request = new StopInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        await _amazonEC2.StopInstancesAsync(request);
        Console.WriteLine("Waiting for the instance to stop.");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Stopped);
    }
}

```

```

        Console.WriteLine("\nThe instance has stopped.");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(
                $"InstanceId is invalid, unable to stop.
{ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while stopping the instance.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{
    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;
    do
    {
        // Wait 5 seconds.
        Thread.Sleep(5000);

        // Check for the desired state.

```



```

        var response = await _amazonEC2.DescribeInstancesAsync(request);
        var instance = response.Reservations[0].Instances[0];
        hasState = instance.State.Name == stateName;
        Console.WriteLine(" ");
    } while (!hasState);

    return hasState;
}

```

- Pour plus de détails sur l'API, reportez-vous [StopInstances](#) à la section Référence des AWS SDK pour .NET API.

TerminateInstances

L'exemple de code suivant montre comment utiliser `TerminateInstances`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Terminate an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance
/// to terminate.</param>
/// <returns>Async task.</returns>
public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
{
    try
    {
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };
    }
}

```

```

        var response = await _amazonEC2.TerminateInstancesAsync(request);
        Console.Write("Waiting for the instance to terminate.");
        await WaitForInstanceState(ec2InstanceId, InstanceStateName.Terminated);

        Console.WriteLine($"\\nThe instance {ec2InstanceId} has been
terminated.");
        return response.TerminatingInstances;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceId")
        {
            _logger.LogError(
                $"InstanceId is invalid, unable to terminate.
{ec2Exception.Message}");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while terminating the instance.: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{
    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;

```

```
do
{
    // Wait 5 seconds.
    Thread.Sleep(5000);

    // Check for the desired state.
    var response = await _amazonEC2.DescribeInstancesAsync(request);
    var instance = response.Reservations[0].Instances[0];
    hasState = instance.State.Name == stateName;
    Console.WriteLine(". ");
} while (!hasState);

return hasState;
}
```

- Pour plus de détails sur l'API, reportez-vous [TerminateInstances](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    // Set up dependency injection for the AWS services.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonDynamoDB>()
                .AddAWSService<IAmazonElasticLoadBalancingV2>()
                .AddAWSService<IAmazonSimpleSystemsManagement>()
                .AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalerWrapper>()
                .AddTransient<ElasticLoadBalancerWrapper>()
                .AddTransient<SmParameterWrapper>()
                .AddTransient<Recommendations>()
                .AddSingleton<IConfiguration>(_configuration)
            )
        .Build();

    ServicesSetup(host);
}
```

```

ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
    await DestroyResources(true);
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>

```

```

    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(

```

```

        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");

```

```

        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

```



```

        await
        _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadBalancerName,
        subnetIds, targetGroup);
        await
        _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
        targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
        _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        var loadBalancerAccess = await
        _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
            that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
            checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
            _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
            ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
                    default VPC must\n"
                    + "allows access from this computer. You can either add it
                    automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
                    \n");

                if (!interactive || GetYesNoResponse(
                    "Do you want to add a rule to the security group to allow
                    inbound traffic from your computer's IP address?"))
                {

```

```

        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
    }
}

if (!sshPortIsOpen)
{
    if (!interactive || GetYesNoResponse(
        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
    {
        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
    }
}

loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

if (loadBalancerAccess)
{
    Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
    Console.WriteLine($"\\thttp://{endPoint}\\n");
}
else
{
    Console.WriteLine(
        "\\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\\n"
        + "manually verifying that your VPC and security group are
configured correctly and that\\n"
        + "you can successfully make a GET request to the load balancer
endpoint:\\n");
    Console.WriteLine($"\\thttp://{endPoint}\\n");
}
Console.WriteLine(new string('-', 80));
Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
if (interactive)
    Console.ReadLine();
return true;
}

```

```

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
        "to create situations where the web service fails, and
shows how using a resilient\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();
}

```

```

        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,

```

```

        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

    Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
    Console.WriteLine("and take that instance out of rotation.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

    Console.WriteLine($"Now, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
    Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
    Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
    Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");

```

```

        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>

```

```

/// <returns>Async task.</returns>
public static async Task<bool> DestroyResources(bool interactive)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
        "that were created for this demo."
    );

    if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
    {
        await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
        await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
        await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
        await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Créez une classe qui englobe les EC2 actions Auto Scaling et Amazon.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,

```



```

    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration,
    ILogger<AutoScalerWrapper> logger)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;
    _logger = logger;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,

```

```

string ssmOnlyPolicyFile,
List<string>? awsManagedPolicies = null)
{

    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", \" +
        "\"Statement\": [{\" +
            "\"Effect\": \"Allow\", \" +
            "\"Principal\": {\" +
            "\"Service\": [\" +
                \"ec2.amazonaws.com\"\" +
            "]" +
            \",\" +
            "\"Action\": \"sts:AssumeRole\"\" +
        "]" +
    "}";

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {
        var createPolicyResult = await _amazonIam.CreatePolicyAsync(
            new CreatePolicyRequest
            {
                PolicyName = policyName,
                PolicyDocument = policyDocument
            });
        policyArn = createPolicyResult.Policy.Arn;
    }
    catch (EntityAlreadyExistsException)
    {
        // The policy already exists, so we look it up to get the Arn.
        var policiesPaginator = _amazonIam.Paginators.ListPolicies(
            new ListPoliciesRequest()
            {
                Scope = PolicyScopeType.Local
            });
        // Get the entire list using the paginator.
        await foreach (var policy in policiesPaginator.Policies)
        {
            if (policy.PolicyName.Equals(policyName))
            {

```

```

        policyArn = policy.Arn;
    }
}

if (policyArn == null)
{
    throw new InvalidOperationException("Policy not found");
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(

```

```

        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });

    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy already exists.");
        var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
            new GetInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)

```

```

        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
    after
    /// the instance is started. This script installs the Python packages and starts
    a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
    run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
    and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
    startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);

```

```

        await CreateInstanceProfileWithName(_instancePolicyName,
            _instanceRoleName,
            _instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes =
            System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
            "InvalidLaunchTemplateName.AlreadyExistsException")
        {
            _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                $"Please try again with a unique name.");
        }

        throw;
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        try
        {
            var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
                new DescribeAvailabilityZonesRequest());
            return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while listing availability zones.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)

```

```

    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        try
        {
            var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
                new DescribeVpcsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("is-default", new List<string>() { "true" })
                    }
                });
            return vpcResponse.Vpcs[0];
        }
    }

```



```

    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "UnauthorizedOperation")
        {
            _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    try
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new("vpc-id", new List<string>() { vpcId }),
                    new("availability-zone", availabilityZones),
                    new("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)

```

```

        {
            subnets.Add(subnet);
        }

        return subnets;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
        {
            _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId} does
not exist.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the subnets.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.NotFoundException")
        {

```

```

        _logger.LogError(
            $"Could not delete the template, the name {_launchTemplateName}
was not found.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
    throw;
}
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                }
            );
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the instance profile.:
{ex.Message}");
        throw;
    }
}

```

```

        });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>

```

```

    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        try
        {
            var response = await
_amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
                new DescribeIamInstanceProfileAssociationsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("instance-id", new List<string>() { instanceId })
                    },
                });
            return response.IamInstanceProfileAssociations[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
                _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>

```

```

    /// <param name="associationId">The Id of the existing profile association for
    the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
    credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
instancesPaginator.InstanceInformationList)
                {
                    instanceReady = instance.InstanceId == instanceId;
                    if (instanceReady)
                    {
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");

```

```

        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                    {
                        "commands",
                        new List<string>() { "cd / && sudo python3 server.py"
80" }
                    }
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while replacing the template.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)

```

```

        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
    progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                    || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        }
    }

```



```

        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
    else
    {
        Console.WriteLine($"No groups found with name {groupName}.");
    }
}

```

```

    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
    computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
    instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;

```

```

        if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
        {
            portIsOpen = true;
        }
    }

    if (ipPermission.PrefixListIds.Any())
    {
        portIsOpen = true;
    }

    if (!portIsOpen)
    {
        Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
    }
    else
    {
        break;
    }
}

return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()

```

```

        {
            new IpPermission()
            {
                FromPort = port,
                ToPort = port,
                IpProtocol = "tcp",
                Ipv4Ranges = new List<IpRange>()
                {
                    new IpRange() { CidrIp = $"{ipAddress}/32" }
                }
            }
        }
    });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
/// <returns>Async task.</returns>
public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
{
    await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
        new AttachLoadBalancerTargetGroupsRequest()
        {
            AutoScalingGroupName = autoScalingGroupName,
            TargetGroupARNs = new List<string>() { targetGroupArn }
        });
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{

```

```

    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;
    do
    {
        // Wait 5 seconds.
        Thread.Sleep(5000);

        // Check for the desired state.
        var response = await _amazonEc2.DescribeInstancesAsync(request);
        var instance = response.Reservations[0].Instances[0];
        hasState = instance.State.Name == stateName;
        Console.Write(". ");
    } while (!hasState);

    return hasState;
}
}

```

Créez une classe qui englobe les actions Elastic Load Balancing.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.

```

```

    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
    }

```

```

        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
    how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///

```

```

    /// To speed up this demo, the health check is configured with shortened times
    and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
    unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
        ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
    subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {

```



```

        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
                    describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
                    LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()

```

```

        {
            Type = ActionTypeEnum.Forward,
            TargetGroupArn = targetGroup.TargetGroupArn
        }
    }
});
return createLbResponse.LoadBalancers[0];
}

/// <summary>
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }
}

```

```

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
                describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try

```

```

        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
            Console.WriteLine($"Deleted load balancing target group
{groupName}.");
            done = true;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine(
                $"Target group {groupName} not found, could not delete.");
            done = true;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine("Target group not yet released, waiting...");
            Thread.Sleep(10000);
        }
    }
}
}

```

Créez une classe qui utilise DynamoDB pour simuler un service de recommandation.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;
}

```

```

public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()

```

```

        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("\nWaiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = tableName
};

TableStatus status;
do
{
    Thread.Sleep(2000);

    var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
}

```

```

        return false;
    }
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}

```

```

    }
}

```

Créez une classe qui englobe les actions de Systems Manager.

```

/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
/// parameters
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }
}

```



```

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
    }
}

```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)

- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Exemples d'Amazon ECS utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon ECS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

ListClusters

L'exemple de code suivant montre comment utiliser `ListClusters`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List cluster ARNs available.
/// </summary>
/// <returns>The ARN list of clusters.</returns>
public async Task<List<string>> GetClusterARNsAsync()
{
    Console.WriteLine("Getting a list of all the clusters in your AWS
account...");
    List<string> clusterArnList = new List<string>();
    // Get a list of all the clusters in your AWS account
    try
    {
        var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
        {
        });

        var clusterArns = listClustersResponse.ClusterArns;

        // Print the ARNs of the clusters
        await foreach (var clusterArn in clusterArns)
        {
            clusterArnList.Add(clusterArn);
        }

        if (clusterArnList.Count == 0)
```

```

        {
            _logger.LogWarning("No clusters found in your AWS account.");
        }
        return clusterArnList;
    }
    catch (Exception e)
    {
        _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListClusters](#) à la section Référence des AWS SDK pour .NET API.

ListServices

L'exemple de code suivant montre comment utiliser `ListServices`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// List service ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of services in given cluster.</returns>
public async Task<List<string>> GetServiceARNsAsync(string clusterARN)
{
    List<string> serviceArns = new List<string>();

    var request = new ListServicesRequest
    {

```

```
        Cluster = clusterARN
    };
    // Call the ListServices API operation and get the list of service ARNs
    var serviceList = _ecsClient.Paginators.ListServices(request);

    await foreach (var serviceARN in serviceList.ServiceArns)
    {
        if (serviceARN is null)
            continue;

        serviceArns.Add(serviceARN);
    }

    if (serviceArns.Count == 0)
    {
        _logger.LogWarning($"No services found in cluster {clusterARN} .");
    }

    return serviceArns;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListServices](#) à la section Référence des AWS SDK pour .NET API.

ListTasks

L'exemple de code suivant montre comment utiliser `ListTasks`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List task ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
```

```
/// <returns>The ARN list of tasks in given cluster.</returns>
public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
{
    // Set up the request to describe the tasks in the service
    var listTasksRequest = new ListTasksRequest
    {
        Cluster = clusterARN
    };
    List<string> taskArns = new List<string>();

    // Call the ListTasks API operation and get the list of task ARNs
    var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

    await foreach (var task in tasks.TaskArns)
    {
        if (task is null)
            continue;

        taskArns.Add(task);
    }

    if (taskArns.Count == 0)
    {
        _logger.LogWarning("No tasks found in cluster: " + clusterARN);
    }

    return taskArns;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListTasks](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Obtention d'informations ARN pour les clusters, les services et les tâches

L'exemple de code suivant illustre comment :

- obtenir une liste de tous les clusters ;
- obtenir des services pour un cluster ;

- obtenir des tâches pour un cluster.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
using Amazon.ECS;
using ECSActions;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSScenario;

public class ECSScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. List ECS Cluster ARNs.
        2. List services in every cluster
        3. List Task ARNs in every cluster.
    */

    private static ILogger logger = null!;
    private static ECSWrapper _ecsWrapper = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
```

```

        logging.AddFilter("System", LogLevel.Debug)
            .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
            .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .Build();

ILoggerFactory loggerFactory = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
});

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<ECSScenario>();

var loggerECSWarpper = LoggerFactory.Create(builder =>
{ builder.AddConsole(); })
    .CreateLogger<ECSWrapper>();

var amazonECSClient = new AmazonECSClient();

_ecsWrapper = new ECSWrapper(amazonECSClient, loggerECSWarpper);

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon ECS example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    await ListClusterARNs();
    await ListServiceARNs();
    await ListTaskARNs();

}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
}
}

/// <summary>
/// List ECS Cluster ARNs
/// </summary>
private static async Task ListClusterARNs()
{

```



```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List Cluster ARNs from ECS.");
        var arns = await _ecsWrapper.GetClusterARNsAsync();

        foreach (var arn in arns)
        {
            Console.WriteLine($"Cluster arn: {arn}");
            Console.WriteLine($"Cluster name: {arn.Split("/").Last()}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List services in every cluster
    /// </summary>
    private static async Task ListServiceARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List Service ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNsAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting services for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var serviceARNs = await _ecsWrapper.GetServiceARNsAsync(clusterARN);

            foreach (var serviceARN in serviceARNs)
            {
                Console.WriteLine($"Service arn: {serviceARN}");
                Console.WriteLine($"Service name: {serviceARN.Split("/").Last()}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>

```

```

    /// List tasks in every cluster
    /// </summary>
    private static async Task ListTaskARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. List Task ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNsAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting tasks for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var taskARNs = await _ecsWrapper.GetTaskARNsAsync(clusterARN);

            foreach (var taskARN in taskARNs)
            {
                Console.WriteLine($"Task arn: {taskARN}");
            }
        }
        Console.WriteLine(new string('-', 80));
    }
}

```

Méthodes d'encapsulation appelées par le scénario pour gérer les actions Amazon ECS.

```

using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.Logging;

namespace ECSActions;

public class ECSWrapper
{
    private readonly AmazonECSClient _ecsClient;
    private readonly ILogger<ECSWrapper> _logger;

    /// <summary>
    /// Constructor for the ECS wrapper.
    /// </summary>
    /// <param name="ecsClient">The injected ECS client.</param>

```

```
/// <param name="logger">The injected logger for the wrapper.</param>
public ECSWrapper(AmazonECSClient ecsClient, ILogger<ECSWrapper> logger)

{
    _logger = logger;
    _ecsClient = ecsClient;
}

/// <summary>
/// List cluster ARNs available.
/// </summary>
/// <returns>The ARN list of clusters.</returns>
public async Task<List<string>> GetClusterARNsAsync()
{
    Console.WriteLine("Getting a list of all the clusters in your AWS
account...");
    List<string> clusterArnList = new List<string>();
    // Get a list of all the clusters in your AWS account
    try
    {
        var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
        {
        });

        var clusterArns = listClustersResponse.ClusterArns;

        // Print the ARNs of the clusters
        await foreach (var clusterArn in clusterArns)
        {
            clusterArnList.Add(clusterArn);
        }

        if (clusterArnList.Count == 0)
        {
            _logger.LogWarning("No clusters found in your AWS account.");
        }
        return clusterArnList;
    }
    catch (Exception e)
    {
    }
}
```

```

        _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
    }
}

/// <summary>
/// List service ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of services in given cluster.</returns>
public async Task<List<string>> GetServiceARNsAsync(string clusterARN)
{
    List<string> serviceArns = new List<string>();

    var request = new ListServicesRequest
    {
        Cluster = clusterARN
    };
    // Call the ListServices API operation and get the list of service ARNs
    var serviceList = _ecsClient.Paginators.ListServices(request);

    await foreach (var serviceARN in serviceList.ServiceArns)
    {
        if (serviceARN is null)
            continue;

        serviceArns.Add(serviceARN);
    }

    if (serviceArns.Count == 0)
    {
        _logger.LogWarning($"No services found in cluster {clusterARN} .");
    }

    return serviceArns;
}

/// <summary>
/// List task ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of tasks in given cluster.</returns>

```

```
public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
{
    // Set up the request to describe the tasks in the service
    var listTasksRequest = new ListTasksRequest
    {
        Cluster = clusterARN
    };
    List<string> taskArns = new List<string>();

    // Call the ListTasks API operation and get the list of task ARNs
    var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

    await foreach (var task in tasks.TaskArns)
    {
        if (task is null)
            continue;

        taskArns.Add(task);
    }

    if (taskArns.Count == 0)
    {
        _logger.LogWarning("No tasks found in cluster: " + clusterARN);
    }

    return taskArns;
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [ListClusters](#)
 - [ListServices](#)
 - [ListTasks](#)

Elastic Load Balancing - Version 2 : exemples d'utilisation SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de la AWS SDK pour .NET version 2 d'Elastic Load Balancing.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

CreateListener

L'exemple de code suivant montre comment utiliser `CreateListener`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
```

```

    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
                    describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
                LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(

```

```

        new CreateListenerRequest()
        {
            LoadBalancerArn = loadBalancerArn,
            Protocol = targetGroup.Protocol,
            Port = targetGroup.Port,
            DefaultActions = new List<Action>()
            {
                new Action()
                {
                    Type = ActionTypeEnum.Forward,
                    TargetGroupArn = targetGroup.TargetGroupArn
                }
            }
        });
    return createLbResponse.LoadBalancers[0];
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateListener](#) à la section Référence des AWS SDK pour .NET API.

CreateLoadBalancer

L'exemple de code suivant montre comment utiliser `CreateLoadBalancer`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create an Elastic Load Balancing load balancer that uses the specified
subnets
/// and forwards requests to the specified target group.
/// </summary>
/// <param name="name">The name for the new load balancer.</param>
/// <param name="subnetIds">Subnets for the load balancer.</param>
/// <param name="targetGroup">Target group for forwarded requests.</param>

```



```

    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
                    describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
                    LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,

```

```

        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
            {
                Type = ActionTypeEnum.Forward,
                TargetGroupArn = targetGroup.TargetGroupArn
            }
        }
    });
    return createLbResponse.LoadBalancers[0];
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateLoadBalancer](#) à la section Référence des AWS SDK pour .NET API.

CreateTargetGroup

L'exemple de code suivant montre comment utiliser `CreateTargetGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
/// requests to instances in the group and how instance health is checked.
///
/// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
/// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
/// </summary>
/// <param name="groupName">The name for the group.</param>
/// <param name="protocol">The protocol, such as HTTP.</param>

```

```
/// <param name="port">The port to use to forward requests, such as 80.</param>
/// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
/// <returns>The new TargetGroup object.</returns>
public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
{
    var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
    var targetGroup = createResponse.TargetGroups[0];
    return targetGroup;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateTargetGroup](#) à la section Référence des AWS SDK pour .NET API.

DeleteLoadBalancer

L'exemple de code suivant montre comment utiliser `DeleteLoadBalancer`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Delete a load balancer by its specified name.
/// </summary>
/// <param name="name">The name of the load balancer to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteLoadBalancerByName(string name)
{
    try
    {
        var describeLoadBalancerResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });
        var lbArn =
            describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
        await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
            new DeleteLoadBalancerRequest()
            {
                LoadBalancerArn = lbArn
            }
        );
    }
    catch (LoadBalancerNotFoundException)
    {
        Console.WriteLine($"Load balancer {name} not found.");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteLoadBalancer](#) à la section Référence des AWS SDK pour .NET API.

DeleteTargetGroup

L'exemple de code suivant montre comment utiliser `DeleteTargetGroup`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
            Console.WriteLine($"Deleted load balancing target group
{groupName}.");
            done = true;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine(
                $"Target group {groupName} not found, could not delete.");
            done = true;
        }
        catch (ResourceInUseException)
        {

```

```

        Console.WriteLine("Target group not yet released, waiting...");
        Thread.Sleep(10000);
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteTargetGroup](#) à la section Référence des AWS SDK pour .NET API.

DescribeLoadBalancers

L'exemple de code suivant montre comment utiliser `DescribeLoadBalancers`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get the HTTP Endpoint of a load balancer by its name.
/// </summary>
/// <param name="loadBalancerName">The name of the load balancer.</param>
/// <returns>The HTTP endpoint.</returns>
public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
{
    if (_endpoint == null)
    {
        var endpointResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { loadBalancerName }
                });
        _endpoint = endpointResponse.LoadBalancers[0].DNSName;
    }
}

```

```
        return _endpoint;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeLoadBalancers](#) à la section Référence des AWS SDK pour .NET API.

DescribeTargetHealth

L'exemple de code suivant montre comment utiliser `DescribeTargetHealth`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the target health for a group by name.
/// </summary>
/// <param name="groupName">The name of the group.</param>
/// <returns>The collection of health descriptions.</returns>
public async Task<List<TargetHealthDescription>>
    CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
```

```
        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
    });
    ;
    result = healthResponse.TargetHealthDescriptions;
}
catch (TargetGroupNotFoundException)
{
    Console.WriteLine($"Target group {groupName} not found.");
}
return result;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeTargetHealth](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    // Set up dependency injection for the AWS services.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonDynamoDB>()
                .AddAWSService<IAmazonElasticLoadBalancingV2>()
                .AddAWSService<IAmazonSimpleSystemsManagement>()
                .AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalerWrapper>()
                .AddTransient<ElasticLoadBalancerWrapper>()
                .AddTransient<SmParameterWrapper>()
                .AddTransient<Recommendations>()
                .AddSingleton<IConfiguration>(_configuration)
            )
        .Build();

    ServicesSetup(host);
}
```

```

        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>

```

```

    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(

```

```

        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");

```

```

        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

```

```

        await
        _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadBalancerName,
        subnetIds, targetGroup);
        await
        _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
        targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
        _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        var loadBalancerAccess = await
        _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
            that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
            checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
            _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
            ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
                    default VPC must\n"
                    + "allows access from this computer. You can either add it
                    automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
                    \n");

                if (!interactive || GetYesNoResponse(
                    "Do you want to add a rule to the security group to allow
                    inbound traffic from your computer's IP address?"))
                {

```

```

        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
    }
}

if (!sshPortIsOpen)
{
    if (!interactive || GetYesNoResponse(
        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
    {
        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
    }
}

loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

if (loadBalancerAccess)
{
    Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
    Console.WriteLine($"http://{endPoint}\n");
}
else
{
    Console.WriteLine(
        "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
        + "manually verifying that your VPC and security group are
configured correctly and that\n"
        + "you can successfully make a GET request to the load balancer
endpoint:\n");
    Console.WriteLine($"http://{endPoint}\n");
}

Console.WriteLine(new string('-', 80));
Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
if (interactive)
    Console.ReadLine();
return true;
}

```

```

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
        "to create situations where the web service fails, and
shows how using a resilient\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();
}

```



```

        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,

```

```

        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

    Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
    Console.WriteLine("and take that instance out of rotation.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

    Console.WriteLine($"Now, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
    Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
    Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
    Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");

```

```

        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>

```

```

/// <returns>Async task.</returns>
public static async Task<bool> DestroyResources(bool interactive)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
        "that were created for this demo."
    );

    if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
    {
        await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
        await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
        await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
        await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Créez une classe qui englobe les EC2 actions Auto Scaling et Amazon.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,

```

```

    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration,
    ILogger<AutoScalerWrapper> logger)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;
    _logger = logger;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,

```

```

string ssmOnlyPolicyFile,
List<string>? awsManagedPolicies = null)
{

    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", \" +
        "\"Statement\": [{\" +
            "\"Effect\": \"Allow\", \" +
            "\"Principal\": {\" +
            "\"Service\": [\" +
                \"ec2.amazonaws.com\"\" +
            "]" +
            \",\" +
            "\"Action\": \"sts:AssumeRole\"\" +
            \"]}\" +
        "}";

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {
        var createPolicyResult = await _amazonIam.CreatePolicyAsync(
            new CreatePolicyRequest
            {
                PolicyName = policyName,
                PolicyDocument = policyDocument
            });
        policyArn = createPolicyResult.Policy.Arn;
    }
    catch (EntityAlreadyExistsException)
    {
        // The policy already exists, so we look it up to get the Arn.
        var policiesPaginator = _amazonIam.Paginators.ListPolicies(
            new ListPoliciesRequest()
            {
                Scope = PolicyScopeType.Local
            });
        // Get the entire list using the paginator.
        await foreach (var policy in policiesPaginator.Policies)
        {
            if (policy.PolicyName.Equals(policyName))
            {

```

```

        policyArn = policy.Arn;
    }
}

if (policyArn == null)
{
    throw new InvalidOperationException("Policy not found");
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(

```



```

        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });

    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy already exists.");
        var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
            new GetInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)

```

```

        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
    after
    /// the instance is started. This script installs the Python packages and starts
    a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
    run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
    and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
    startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);

```

```

        await CreateInstanceProfileWithName(_instancePolicyName,
        _instanceRoleName,
            _instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes =
        System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new

LaunchTemplateIamInstanceProfileSpecificationRequest()
                {
                    Name = _instanceProfileName
                },
                KeyName = _keyPairName,
                UserData = System.Convert.ToBase64String(plainTextBytes)
            }
        });
        return launchTemplateResponse.LaunchTemplate;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
        "InvalidLaunchTemplateName.AlreadyExistsException")
        {
            _logger.LogError($"Could not create the template, the name
        {_launchTemplateName} already exists. " +
                $"Please try again with a unique name.");
        }

        throw;
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        try
        {
            var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
                new DescribeAvailabilityZonesRequest());
            return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while listing availability zones.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)

```

```

    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        try
        {
            var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
                new DescribeVpcsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("is-default", new List<string>() { "true" })
                    }
                });
            return vpcResponse.Vpcs[0];
        }
    }

```

```

    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "UnauthorizedOperation")
        {
            _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    try
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new("vpc-id", new List<string>() { vpcId }),
                    new("availability-zone", availabilityZones),
                    new("default-for-az", new List<string>() { "true" })
                }
            }
        );

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)

```

```

        {
            subnets.Add(subnet);
        }

        return subnets;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
        {
            _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId} does
not exist.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the subnets.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.NotFoundException")
        {

```

```

        _logger.LogError(
            $"Could not delete the template, the name {_launchTemplateName}
was not found.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
    throw;
}
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                }
            );
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the instance profile.:
{ex.Message}");
        throw;
    }
}

```



```

        });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>

```

```

    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        try
        {
            var response = await
_amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
                new DescribeIamInstanceProfileAssociationsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("instance-id", new List<string>() { instanceId })
                    },
                });
            return response.IamInstanceProfileAssociations[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
                _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>

```

```

    /// <param name="associationId">The Id of the existing profile association for
    the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
    credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
instancesPaginator.InstanceInformationList)
                {
                    instanceReady = instance.InstanceId == instanceId;
                    if (instanceReady)
                    {
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");

```

```

        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                    {
                        "commands",
                        new List<string>() { "cd / && sudo python3 server.py"
80" }
                    }
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while replacing the template.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)

```

```

        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
    progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                    || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        }
    }

```

```

        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
    else
    {
        Console.WriteLine($"No groups found with name {groupName}.");
    }
}

```

```

    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
    computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
    instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
    ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;

```

```

        if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
        {
            portIsOpen = true;
        }
    }

    if (ipPermission.PrefixListIds.Any())
    {
        portIsOpen = true;
    }

    if (!portIsOpen)
    {
        Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
    }
    else
    {
        break;
    }
}

return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()

```



```

        {
            new IpPermission()
            {
                FromPort = port,
                ToPort = port,
                IpProtocol = "tcp",
                Ipv4Ranges = new List<IpRange>()
                {
                    new IpRange() { CidrIp = $"{ipAddress}/32" }
                }
            }
        }
    });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
/// <returns>Async task.</returns>
public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
{
    await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
        new AttachLoadBalancerTargetGroupsRequest()
        {
            AutoScalingGroupName = autoScalingGroupName,
            TargetGroupARNs = new List<string>() { targetGroupArn }
        });
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{

```

```

    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;
    do
    {
        // Wait 5 seconds.
        Thread.Sleep(5000);

        // Check for the desired state.
        var response = await _amazonEc2.DescribeInstancesAsync(request);
        var instance = response.Reservations[0].Instances[0];
        hasState = instance.State.Name == stateName;
        Console.Write(". ");
    } while (!hasState);

    return hasState;
}
}

```

Créez une classe qui englobe les actions Elastic Load Balancing.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.

```

```

    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
    }

```

```

        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
    how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///

```

```

    /// To speed up this demo, the health check is configured with shortened times
    and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
    unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
        ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
    subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {

```

```

        var createLbResponse = await
        _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
                    describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
                    LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()

```

```

        {
            Type = ActionTypeEnum.Forward,
            TargetGroupArn = targetGroup.TargetGroupArn
        }
    }
});
return createLbResponse.LoadBalancers[0];
}

/// <summary>
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }
}

```

```

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
                describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try

```



```

        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
            Console.WriteLine($"Deleted load balancing target group
{groupName}.");
            done = true;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine(
                $"Target group {groupName} not found, could not delete.");
            done = true;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine("Target group not yet released, waiting...");
            Thread.Sleep(10000);
        }
    }
}
}

```

Créez une classe qui utilise DynamoDB pour simuler un service de recommandation.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;
}

```

```

public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()

```

```

        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("\nWaiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = tableName
};

TableStatus status;
do
{
    Thread.Sleep(2000);

    var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
}

```

```

        return false;
    }
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}

```

```

    }
}

```

Créez une classe qui englobe les actions de Systems Manager.

```

/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
/// parameters
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }
}

```

```

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
    }
}

```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)

- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

EventBridge exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with EventBridge.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour EventBridge

L'exemple de code suivant montre comment démarrer avec EventBridge.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.EventBridge;
using Amazon.EventBridge.Model;

namespace EventBridgeActions;

public static class HelloEventBridge
{
    static async Task Main(string[] args)
    {
        var eventBridgeClient = new AmazonEventBridgeClient();

        Console.WriteLine($"Hello Amazon EventBridge! Following are some of your
EventBuses:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five event buses.
        var response = await eventBridgeClient.ListEventBusesAsync(
            new ListEventBusesRequest()
            {
                Limit = 5
            });

        foreach (var eventBus in response.EventBuses)
        {
            Console.WriteLine($"\\tEventBus: {eventBus.Name}");
            Console.WriteLine($"\\tArn: {eventBus.Arn}");
        }
    }
}
```



```
        Console.WriteLine($"\\tPolicy: {eventBus.Policy}");  
        Console.WriteLine();  
    }  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [ListEventBuses](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez une règle et ajoutez-y une cible.
- Activez et désactivez les règles.
- Répertoriez et mettez à jour les règles et les cibles.
- Envoyez des événements, puis nettoyez les ressources.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
public class EventBridgeScenario  
{  
    /*  
        Before running this .NET code example, set up your development environment,  
        including your credentials.  
  
        This .NET example performs the following tasks with Amazon EventBridge:  
        - Create a rule.
```

- Add a target to a rule.
- Enable and disable rules.
- List rules and targets.
- Update rules and targets.
- Send events.
- Delete the rule.

*/

```
private static ILogger logger = null!;
private static EventBridgeWrapper _eventBridgeWrapper = null!;
private static IConfiguration _configuration = null!;

private static IAmazonIdentityManagementService? _iamClient = null!;
private static IAmazonSimpleNotificationService? _snsClient = null!;
private static IAmazonS3 _s3Client = null!;

static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EventBridge.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonEventBridge>()
                .AddAWSService<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonS3>()
                .AddAWSService<IAmazonSimpleNotificationService>()
                .AddTransient<EventBridgeWrapper>()
            )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<EventBridgeScenario>();
```

```
ServicesSetup(host);

string topicArn = "";
string roleArn = "";

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon EventBridge example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    roleArn = await CreateRole();

    await CreateBucketWithEventBridgeEvents();

    await AddEventRule(roleArn);

    await ListEventRules();

    topicArn = await CreateSnsTopic();

    var email = await SubscribeToSnsTopic(topicArn);

    await AddSnsTarget(topicArn);

    await ListTargets();

    await ListRulesForTarget(topicArn);

    await UploadS3File(_s3Client);

    await ChangeRuleState(false);

    await GetRuleState();

    await UpdateSnsEventRule(topicArn);

    await ChangeRuleState(true);

    await UploadS3File(_s3Client);

    await UpdateToCustomRule(topicArn);

    await TriggerCustomRule(email);
```

```

        await CleanupResources(topicArn);
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
        await CleanupResources(topicArn);
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("The Amazon EventBridge example scenario is complete.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _eventBridgeWrapper =
host.Services.GetRequiredService<EventBridgeWrapper>();
    _snsClient =
host.Services.GetRequiredService<IAmazonSimpleNotificationService>();
    _s3Client = host.Services.GetRequiredService<IAmazonS3>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
}

/// <summary>
/// Create a role to be used by EventBridge.
/// </summary>
/// <returns>The role Amazon Resource Name (ARN).</returns>
public static async Task<string> CreateRole()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating a role to use with EventBridge and attaching
managed policy AmazonEventBridgeFullAccess.");
    Console.WriteLine(new string('-', 80));

    var roleName = _configuration["roleName"];

    var assumeRolePolicy = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +

```

```

        "\"Effect\": \"Allow\", \" +
        "\"Principal\": {\" +
        $\"\"Service\": \"events.amazonaws.com\"\" +
        \"},\" +
        "\"Action\": \"sts:AssumeRole\"\" +
        \"}]\" +
        \"}\";

var roleResult = await _iamClient!.CreateRoleAsync(
    new CreateRoleRequest()
    {
        AssumeRolePolicyDocument = assumeRolePolicy,
        Path = "/",
        RoleName = roleName
    });

await _iamClient.AttachRolePolicyAsync(
    new AttachRolePolicyRequest()
    {
        PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
        RoleName = roleName
    });
// Allow time for the role to be ready.
Thread.Sleep(10000);
return roleResult.Role.Arn;
}

/// <summary>
/// Create an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge
events enabled.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CreateBucketWithEventBridgeEvents()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating an S3 bucket with EventBridge events enabled.");

    var testBucketName = _configuration["testBucketName"];

    var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
    testBucketName);

    if (!bucketExists)

```

```

    {
        await _s3Client.PutBucketAsync(new PutBucketRequest()
        {
            BucketName = testBucketName,
            UseClientRegion = true
        });
    }

    await _s3Client.PutBucketNotificationAsync(new
PutBucketNotificationRequest()
    {
        BucketName = testBucketName,
        EventBridgeConfiguration = new EventBridgeConfiguration()
    });

    Console.WriteLine($"\\tAdded bucket {testBucketName} with EventBridge events
enabled.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create and upload a file to an S3 bucket to trigger an event.
/// </summary>
/// <returns>Async task.</returns>
private static async Task UploadS3File(IAmazonS3 s3Client)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Uploading a file to the test bucket. This will trigger a
subscription email.");

    var testBucketName = _configuration["testBucketName"];

    var fileName = $"example_upload_{DateTime.UtcNow.Ticks}.txt";

    // Create the file if it does not already exist.
    if (!File.Exists(fileName))
    {
        await using StreamWriter sw = File.CreateText(fileName);
        await sw.WriteLineAsync(
            "This is a sample file for testing uploads.");
    }

    await s3Client.PutObjectAsync(new PutObjectRequest()

```

```

    {
        FilePath = fileName,
        BucketName = testBucketName
    });

    Console.WriteLine($"\\tPress Enter to continue.");
    Console.ReadLine();

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create an Amazon Simple Notification Service (Amazon SNS) topic to use as an
EventBridge target.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string> CreateSnsTopic()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "Creating an Amazon Simple Notification Service (Amazon SNS) topic for
email subscriptions.");

    var topicName = _configuration["topicName"];

    string topicPolicy = "{" +
        "\\\"Version\\\": \\\"2012-10-17\\\", \" +
        "\\\"Statement\\\": [{\" +
        "\\\"Sid\\\": \\\"EventBridgePublishTopic\\\", \" +
        "\\\"Effect\\\": \\\"Allow\\\", \" +
        "\\\"Principal\\\": {\" +
        $\"\\\"Service\\\": \\\"events.amazonaws.com\\\"\" +
        \"}, \" +
        "\\\"Resource\\\": \\\"*\\\", \" +
        "\\\"Action\\\": \\\"sns:Publish\\\"\" +
        \"}]}\" +
        \"\";

    var topicAttributes = new Dictionary<string, string>()
    {
        { \"Policy\", topicPolicy }
    };
};

```

```

        var topicResponse = await _snsClient!.CreateTopicAsync(new
CreateTopicRequest()
    {
        Name = topicName,
        Attributes = topicAttributes

    });

    Console.WriteLine($"\\tAdded topic {topicName} for email subscriptions.");

    Console.WriteLine(new string('-', 80));

    return topicResponse.TopicArn;
}

/// <summary>
/// Subscribe a user email to an SNS topic.
/// </summary>
/// <param name="topicArn">The ARN of the SNS topic.</param>
/// <returns>The user's email.</returns>
private static async Task<string> SubscribeToSnsTopic(string topicArn)
{
    Console.WriteLine(new string('-', 80));

    string email = "";
    while (string.IsNullOrEmpty(email))
    {
        Console.WriteLine("Enter your email to subscribe to the Amazon SNS
topic:");
        email = Console.ReadLine()!;
    }

    var subscriptions = new List<string>();
    var paginatedSubscriptions =
_snsClient!.Paginators.ListSubscriptionsByTopic(
    new ListSubscriptionsByTopicRequest()
    {
        TopicArn = topicArn
    });

    // Get the entire list using the paginator.
    await foreach (var subscription in paginatedSubscriptions.Subscriptions)
    {

```



```

        subscriptions.Add(subscription.Endpoint);
    }

    if (subscriptions.Contains(email))
    {
        Console.WriteLine($"\\tYour email is already subscribed.");
        Console.WriteLine(new string('-', 80));
        return email;
    }

    await _snsClient.SubscribeAsync(new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "email",
        Endpoint = email
    });

    Console.WriteLine($"Use the link in the email you received to confirm your
subscription, then press Enter to continue.");

    Console.ReadLine();

    Console.WriteLine(new string('-', 80));
    return email;
}

/// <summary>
/// Add a rule which triggers when a file is uploaded to an S3 bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role used by EventBridge.</param>
/// <returns>Async task.</returns>
private static async Task AddEventRule(string roleArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Creating an EventBridge event that sends an email when an
Amazon S3 object is created.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];

    await _eventBridgeWrapper.PutS3UploadRule(roleArn, eventRuleName,
testBucketName);
    Console.WriteLine($"\\tAdded event rule {eventRuleName} for bucket
{testBucketName}.");
}

```

```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an SNS target to the rule.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
    private static async Task AddSnsTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Adding a target to the rule to that sends an email when
the rule is triggered.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];
        var topicName = _configuration["topicName"];
        await _eventBridgeWrapper.AddSnsTargetToRule(eventRuleName, topicArn);
        Console.WriteLine($"\\tAdded event rule {eventRuleName} with Amazon SNS
target {topicName} for bucket {testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the event rules on the default event bus.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListEventRules()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Current event rules:");

        var rules = await _eventBridgeWrapper.ListAllRulesForEventBus();
        rules.ForEach(r => Console.WriteLine($"\\tRule: {r.Name} Description:
{r.Description} State: {r.State}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Update the event target to use a transform.
    /// </summary>

```

```

/// <param name="topicArn">The SNS topic ARN target to update.</param>
/// <returns>Async task.</returns>
private static async Task UpdateSnsEventRule(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Let's update the event target with a transform.");

    var eventRuleName = _configuration["eventRuleName"];
    var testBucketName = _configuration["testBucketName"];

    await
_eventBridgeWrapper.UpdateS3UploadRuleTargetWithTransform(eventRuleName, topicArn);
    Console.WriteLine($"\\tUpdated event rule {eventRuleName} with Amazon SNS
target {topicArn} for bucket {testBucketName}.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Update the rule to use a custom event pattern.
/// </summary>
/// <returns>Async task.</returns>
private static async Task UpdateToCustomRule(string topicArn)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Updating the event pattern to be triggered by a custom
event instead.");

    var eventRuleName = _configuration["eventRuleName"];

    await _eventBridgeWrapper.UpdateCustomEventPattern(eventRuleName);

    Console.WriteLine($"\\tUpdated event rule {eventRuleName} to custom
pattern.");
    await _eventBridgeWrapper.UpdateCustomRuleTargetWithTransform(eventRuleName,
topicArn);

    Console.WriteLine($"\\tUpdated event target {topicArn}.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Send rule events for a custom rule using the user's email address.

```

```

    /// </summary>
    /// <param name="email">The email address to include.</param>
    /// <returns>Async task.</returns>
    private static async Task TriggerCustomRule(string email)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Sending an event to trigger the rule. This will trigger a
subscription email.");

        await _eventBridgeWrapper.PutCustomEmailEvent(email);

        Console.WriteLine($"\\tEvents have been sent. Press Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the targets for a rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListTargets()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the targets for a particular rule.");

        var eventRuleName = _configuration["eventRuleName"];
        var targets = await _eventBridgeWrapper.ListAllTargetsOnRule(eventRuleName);
        targets.ForEach(t => Console.WriteLine($"\\tTarget: {t.Arn} Id: {t.Id} Input:
{t.Input}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the rules for a particular target.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
    private static async Task ListRulesForTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the rules for a particular target.");
    }

```

```

        var rules = await _eventBridgeWrapper.ListAllRuleNamesByTarget(topicArn);
        rules.ForEach(r => Console.WriteLine($"\\tRule: {r}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Enable or disable a particular rule.
    /// </summary>
    /// <param name="isEnabled">True to enable the rule, otherwise false.</param>
    /// <returns>Async task.</returns>
    private static async Task ChangeRuleState(bool isEnabled)
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];

        if (!isEnabled)
        {
            Console.WriteLine($"Disabling the rule: {eventRuleName}");
            await _eventBridgeWrapper.DisableRuleByName(eventRuleName);
        }
        else
        {
            Console.WriteLine($"Enabling the rule: {eventRuleName}");
            await _eventBridgeWrapper.EnableRuleByName(eventRuleName);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the current state of the rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetRuleState()
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];

        var state = await _eventBridgeWrapper.GetRuleStateByRuleName(eventRuleName);
        Console.WriteLine($"Rule {eventRuleName} is in current state {state}.");

        Console.WriteLine(new string('-', 80));
    }
}

```

```

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic to clean up.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        var eventRuleName = _configuration["eventRuleName"];
        if (GetYesNoResponse($"\\tDelete all targets and event rule {eventRuleName}?
(y/n)"))
        {
            Console.WriteLine($"\\tRemoving all targets from the event rule.");
            await _eventBridgeWrapper.RemoveAllTargetsFromRule(eventRuleName);

            Console.WriteLine($"\\tDeleting event rule.");
            await _eventBridgeWrapper.DeleteRuleByName(eventRuleName);
        }

        var topicName = _configuration["topicName"];
        if (GetYesNoResponse($"\\tDelete Amazon SNS subscription topic {topicName}?
(y/n)"))
        {
            Console.WriteLine($"\\tDeleting topic.");
            await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        }

        var bucketName = _configuration["testBucketName"];
        if (GetYesNoResponse($"\\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
        {
            Console.WriteLine($"\\tDeleting bucket.");
            // Delete all objects in the bucket.
            var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()

```

```

        {
            BucketName = bucketName,
            Objects = deleteList.S3Objects
                .Select(o => new KeyVersion { Key = o.Key }).ToList()
        });
        // Now delete the bucket.
        await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
        {
            BucketName = bucketName
        });
    }

    var roleName = _configuration["roleName"];
    if (GetYesNoResponse($"\\tDelete role {roleName}? (y/n)"))
    {
        Console.WriteLine($"\\tDetaching policy and deleting role.");

        await _iamClient!.DetachRolePolicyAsync(new DetachRolePolicyRequest()
        {
            RoleName = roleName,
            PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
        });

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = roleName
        });
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null &&
        ynResponse.Equals("y",
            StringComparison.InvariantCultureIgnoreCase);
}

```

```

        return response;
    }
}

```

Créez une classe qui englobe les EventBridge opérations.

```

/// <summary>
/// Wrapper for Amazon EventBridge operations.
/// </summary>
public class EventBridgeWrapper
{
    private readonly IAmazonEventBridge _amazonEventBridge;
    private readonly ILogger<EventBridgeWrapper> _logger;

    /// <summary>
    /// Constructor for the EventBridge wrapper.
    /// </summary>
    /// <param name="amazonEventBridge">The injected EventBridge client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public EventBridgeWrapper(IAmazonEventBridge amazonEventBridge,
        ILogger<EventBridgeWrapper> logger)

    {
        _amazonEventBridge = amazonEventBridge;
        _logger = logger;
    }

    /// <summary>
    /// Get the state for a rule by the rule name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="eventBusName">The optional name of the event bus. If empty,
    uses the default event bus.</param>
    /// <returns>The state of the rule.</returns>
    public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
    {
        var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
            new DescribeRuleRequest()
            {
                Name = ruleName,

```



```

        EventBusName = eventBusName
    });
    return ruleResponse.State;
}

/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
        new EnableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Disable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
{

```

```

        var results = new List<Rule>();
        var request = new ListRulesRequest()
        {
            EventBusName = eventBusArn
        };
        // Get all of the pages of rules.
        ListRulesResponse response;
        do
        {
            response = await _amazonEventBridge.ListRulesAsync(request);
            results.AddRange(response.Rules);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// List all of the targets matching a rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>The list of targets.</returns>
    public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
    {
        var results = new List<Target>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse response;
        do
        {
            response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
            results.AddRange(response.Targets);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// List names of all rules matching a target.

```

```

    /// </summary>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <returns>The list of rule names.</returns>
    public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
    {
        var results = new List<string>();
        var request = new ListRuleNamesByTargetRequest()
        {
            TargetArn = targetArn
        };
        ListRuleNamesByTargetResponse response;
        do
        {
            response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
            results.AddRange(response.RuleNames);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// Create a new event rule that triggers when an Amazon S3 object is created in
    a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
string bucketName)
    {
        string eventPattern = "{" +
            "\"source\": [\"aws.s3\"],\" +
            "\"detail-type\": [\"Object Created\"],\" +
            "\"detail\": {\" +
            \"bucket\": {\" +
            \"name\": [\"\" + bucketName + "\"]\" +
            \"}\" +
            \"}\" +
            \"}\";
    }

```

```

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
    event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
        string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
                InputTransformer = new InputTransformer()
                {
                    InputPathsMap = new Dictionary<string, string>()
                    {
                        {"bucket", "$.detail.bucket.name"},
                        {"time", "$.time"}
                    },
                    InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\\""
                }
            }
        };

        var response = await _amazonEventBridge.PutTargetsAsync(

```

```

        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });
    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }
    return targetID;
}

/// <summary>
/// Update a custom rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateCustomRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                InputTemplate = "\"Notification: sample event was received.\"\""
            }
        }
    };

    var response = await _amazonEventBridge.PutTargetsAsync(

```

```

        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });
    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }
    return targetID;
}

/// <summary>
/// Add an event to the event bus that includes an email, message, and time.
/// </summary>
/// <param name="email">The email to use in the event detail of the custom
event.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutCustomEmailEvent(string email)
{
    var eventDetail = new
    {
        UserEmail = email,
        Message = "This event was generated by example code.",
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
                {
                    Source = "ExampleSource",
                    Detail = JsonSerializer.Serialize(eventDetail),
                    DetailType = "ExampleType"
                }
            }
        }
    );
}

```

```

    });

    return response.FailedEntryCount == 0;
}

/// <summary>
/// Update a rule to use a custom defined event pattern.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <returns>The ARN of the updated rule.</returns>
public async Task<string> UpdateCustomEventPattern(string ruleName)
{
    string customEventsPattern = "{" +
                                   "\"source\": [\"ExampleSource\"],\" +
                                   "\"detail-type\": [\"ExampleType\"]\" +
                                   "}";

    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Custom test rule",
            EventPattern = customEventsPattern
        });

    return response.RuleArn;
}

/// <summary>
/// Add an Amazon SNS target topic to a rule.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <param name="targetArn">The ARN of the Amazon SNS target.</param>
/// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    // Create the list of targets and add a new target.
    var targets = new List<Target>
    {

```

```

        new Target()
        {
            Arn = targetArn,
            Id = targetID
        }
    };

    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
            Targets = targets,
        });

    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }

    return targetID;
}

/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
{
    var targetIds = new List<string>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse targetsResponse;
    do
    {

```



```

        targetsResponse = await
        _amazonEventBridge.ListTargetsByRuleAsync(request);
        targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
        request.NextToken = targetsResponse.NextToken;

    } while (targetsResponse.NextToken is not null);

    var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
        new RemoveTargetsRequest()
        {
            Rule = ruleName,
            Ids = targetIds
        });

    if (removeResponse.FailedEntryCount > 0)
    {
        removeResponse.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }

    return removeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
}

```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [DeleteRule](#)
 - [DescribeRule](#)
 - [DisableRule](#)
 - [EnableRule](#)
 - [ListRuleNamesByTarget](#)
 - [ListRules](#)
 - [ListTargetsByRule](#)
 - [PutEvents](#)
 - [PutRule](#)
 - [PutTargets](#)

Actions

DeleteRule

L'exemple de code suivant montre comment utiliser `DeleteRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Supprimez une règle par son nom.

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
```

```
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteRule](#) à la section Référence des AWS SDK pour .NET API.

DescribeRule

L'exemple de code suivant montre comment utiliser `DescribeRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Obtenez l'état d'une règle à l'aide de sa description.

```
/// <summary>
/// Get the state for a rule by the rule name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
/// <returns>The state of the rule.</returns>
public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
{
    var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
        new DescribeRuleRequest()
        {
            Name = ruleName,
```

```

        EventBusName = eventBusName
    });
    return ruleResponse.State;
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeRule](#) à la section Référence des AWS SDK pour .NET API.

DisableRule

L'exemple de code suivant montre comment utiliser `DisableRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Désactivez une règle par son nom.

```

/// <summary>
/// Disable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}

```

- Pour plus de détails sur l'API, reportez-vous [DisableRule](#) à la section Référence des AWS SDK pour .NET API.

EnableRule

L'exemple de code suivant montre comment utiliser `EnableRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Activez une règle par son nom.

```
/// <summary>
/// Enable a particular rule on an event bus.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
        new EnableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [EnableRule](#) à la section Référence des AWS SDK pour .NET API.

ListRuleNamesByTarget

L'exemple de code suivant montre comment utiliser `ListRuleNamesByTarget`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Répertoriez tous les noms de règle à l'aide de la cible.

```
/// <summary>
/// List names of all rules matching a target.
/// </summary>
/// <param name="targetArn">The ARN of the target.</param>
/// <returns>The list of rule names.</returns>
public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
{
    var results = new List<string>();
    var request = new ListRuleNamesByTargetRequest()
    {
        TargetArn = targetArn
    };
    ListRuleNamesByTargetResponse response;
    do
    {
        response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
        results.AddRange(response.RuleNames);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListRuleNamesByTarget](#) à la section Référence des AWS SDK pour .NET API.

ListRules

L'exemple de code suivant montre comment utiliser `ListRules`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Répertoriez toutes les règles pour un bus d'événements.

```
/// <summary>
/// List the rules on an event bus.
/// </summary>
/// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
/// <returns>The list of rules.</returns>
public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
{
    var results = new List<Rule>();
    var request = new ListRulesRequest()
    {
        EventBusName = eventBusArn
    };
    // Get all of the pages of rules.
    ListRulesResponse response;
    do
    {
        response = await _amazonEventBridge.ListRulesAsync(request);
        results.AddRange(response.Rules);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListRules](#) à la section Référence des AWS SDK pour .NET API.

ListTargetsByRule

L'exemple de code suivant montre comment utiliser `ListTargetsByRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Répertoriez toutes les cibles d'une règle à l'aide de son nom.

```
/// <summary>
/// List all of the targets matching a rule by name.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <returns>The list of targets.</returns>
public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
{
    var results = new List<Target>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse response;
    do
    {
        response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
        results.AddRange(response.Targets);
        request.NextToken = response.NextToken;
    } while (response.NextToken is not null);

    return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListTargetsByRule](#) à la section Référence des AWS SDK pour .NET API.

PutEvents

L'exemple de code suivant montre comment utiliser PutEvents.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Envoyez un événement qui correspond à un modèle personnalisé pour une règle.

```
/// <summary>
/// Add an event to the event bus that includes an email, message, and time.
/// </summary>
/// <param name="email">The email to use in the event detail of the custom
event.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutCustomEmailEvent(string email)
{
    var eventDetail = new
    {
        UserEmail = email,
        Message = "This event was generated by example code.",
        UtcTime = DateTime.UtcNow.ToString("g")
    };
    var response = await _amazonEventBridge.PutEventsAsync(
        new PutEventsRequest()
        {
            Entries = new List<PutEventsRequestEntry>()
            {
                new PutEventsRequestEntry()
                {
                    Source = "ExampleSource",
                    Detail = JsonSerializer.Serialize(eventDetail),
                    DetailType = "ExampleType"
                }
            }
        });

    return response.FailedEntryCount == 0;
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [PutEvents](#) à la section Référence des AWS SDK pour .NET API.

PutRule

L'exemple de code suivant montre comment utiliser `PutRule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une règle qui se déclenche lorsqu'un objet est ajouté à un compartiment Amazon Simple Storage Service.

```
/// <summary>
/// Create a new event rule that triggers when an Amazon S3 object is created in
a bucket.
/// </summary>
/// <param name="roleArn">The ARN of the role.</param>
/// <param name="ruleName">The name to give the rule.</param>
/// <param name="bucketName">The name of the bucket to trigger the event.</
param>
/// <returns>The ARN of the new rule.</returns>
public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
string bucketName)
{
    string eventPattern = "{" +
        "\"source\": [\"aws.s3\"],\" +
        "\"detail-type\": [\"Object Created\"],\" +
        "\"detail\": {\" +
            \"bucket\": {\" +
                \"name\": [\"" + bucketName + "\"]\" +
            }\" +
        }\" +
    }
```

```

        "});

    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Example S3 upload rule for EventBridge",
            RoleArn = roleArn,
            EventPattern = eventPattern
        });

    return response.RuleArn;
}

```

Créez une règle qui utilise un modèle personnalisé.

```

/// <summary>
/// Update a rule to use a custom defined event pattern.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <returns>The ARN of the updated rule.</returns>
public async Task<string> UpdateCustomEventPattern(string ruleName)
{
    string customEventsPattern = "{" +
                                "\"source\": [\"ExampleSource\"],\" +
                                "\"detail-type\": [\"ExampleType\"]\" +
                                "}";

    var response = await _amazonEventBridge.PutRuleAsync(
        new PutRuleRequest()
        {
            Name = ruleName,
            Description = "Custom test rule",
            EventPattern = customEventsPattern
        });

    return response.RuleArn;
}

```

- Pour plus de détails sur l'API, reportez-vous [PutRule](#) à la section Référence des AWS SDK pour .NET API.

PutTargets

L'exemple de code suivant montre comment utiliser `PutTargets`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Ajoutez une rubrique Amazon SNS en tant que cible pour une règle.

```
/// <summary>
/// Add an Amazon SNS target topic to a rule.
/// </summary>
/// <param name="ruleName">The name of the rule to update.</param>
/// <param name="targetArn">The ARN of the Amazon SNS target.</param>
/// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    // Create the list of targets and add a new target.
    var targets = new List<Target>
    {
        new Target()
        {
            Arn = targetArn,
            Id = targetID
        }
    };

    // Add the targets to the rule.
    var response = await _amazonEventBridge.PutTargetsAsync(
        new PutTargetsRequest()
        {
            EventBusName = eventBusArn,
            Rule = ruleName,
```

```

        Targets = targets,
    });

    if (response.FailedEntryCount > 0)
    {
        response.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }

    return targetID;
}

```

Ajoutez un transformateur d'entrée à une cible pour une règle.

```

/// <summary>
/// Update an Amazon S3 object created rule with a transform on the target.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="targetArn">The ARN of the target.</param>
/// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
/// <returns>The ID of the target.</returns>
public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
{
    var targetID = Guid.NewGuid().ToString();

    var targets = new List<Target>
    {
        new Target()
        {
            Id = targetID,
            Arn = targetArn,
            InputTransformer = new InputTransformer()
            {
                InputPathsMap = new Dictionary<string, string>()
                {
                    {"bucket", "$.detail.bucket.name"},

```

```

        {"time", "${.time}"
    },
    InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\\""
    }
}
};
var response = await _amazonEventBridge.PutTargetsAsync(
    new PutTargetsRequest()
    {
        EventBusName = eventBusArn,
        Rule = ruleName,
        Targets = targets,
    });
if (response.FailedEntryCount > 0)
{
    response.FailedEntries.ForEach(e =>
    {
        _logger.LogError(
            $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
    });
}
return targetID;
}

```

- Pour plus de détails sur l'API, reportez-vous [PutTargets](#) à la section Référence des AWS SDK pour .NET API.

RemoveTargets

L'exemple de code suivant montre comment utiliser `RemoveTargets`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Retirez toutes les cibles d'une règle à l'aide de son nom.

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
{
    var targetIds = new List<string>();
    var request = new ListTargetsByRuleRequest()
    {
        Rule = ruleName
    };
    ListTargetsByRuleResponse targetsResponse;
    do
    {
        targetsResponse = await
            _amazonEventBridge.ListTargetsByRuleAsync(request);
        targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
        request.NextToken = targetsResponse.NextToken;
    } while (targetsResponse.NextToken is not null);

    var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
        new RemoveTargetsRequest()
        {
            Rule = ruleName,
            Ids = targetIds
        });

    if (removeResponse.FailedEntryCount > 0)
    {
        removeResponse.FailedEntries.ForEach(e =>
        {
            _logger.LogError(
                $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
        });
    }

    return removeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [RemoveTargets](#) à la section Référence des AWS SDK pour .NET API.

EventBridge Exemples de planificateur utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET EventBridge planificateur with.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Actions](#)
- [Scénarios](#)

Mise en route

Bonjour EventBridge Scheduler

L'exemple de code suivant montre comment commencer à utiliser EventBridge Scheduler.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).


```

public static class HelloScheduler
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the EventBridge Scheduler service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonScheduler>()
            ).Build();

        // Now the client is available for injection.
        var schedulerClient = host.Services.GetRequiredService<IAmazonScheduler>();

        // You can use await and any of the async methods to get a response, or a
        paginator to list schedules or groups.
        var results = new List<ScheduleSummary>();
        var paginateSchedules = schedulerClient.Paginators.ListSchedules(
            new ListSchedulesRequest());
        Console.WriteLine(
            $"Hello AWS Scheduler! Let's list schedules in your account.");
        // Get the entire list using the paginator.
        await foreach (var schedule in paginateSchedules.Schedules)
        {
            results.Add(schedule);
        }
        Console.WriteLine($"Total of {results.Count} schedule(s) available.");
        results.ForEach(s => Console.WriteLine($"Schedule: {s.Name}"));
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListSchedules](#) à la section Référence des AWS SDK pour .NET API.

Actions

CreateSchedule

L'exemple de code suivant montre comment utiliser `CreateSchedule`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Creates a new schedule in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule.</param>
    /// <param name="scheduleExpression">The schedule expression that defines when
the schedule should run.</param>
    /// <param name="scheduleGroupName">The name of the schedule group to which the
schedule should be added.</param>
    /// <param name="deleteAfterCompletion">Indicates whether to delete the schedule
after completion.</param>
    /// <param name="useFlexibleTimeWindow">Indicates whether to use a flexible time
window for the schedule.</param>
    /// <param name="targetArn">ARN of the event target.</param>
    /// <param name="roleArn">Execution Role ARN.</param>
    /// <returns>True if the schedule was created successfully, false otherwise.</
returns>
    public async Task<bool> CreateScheduleAsync(
        string name,
        string scheduleExpression,
        string scheduleGroupName,
        string targetArn,
        string roleArn,
        string input,
        bool deleteAfterCompletion = false,
        bool useFlexibleTimeWindow = false)
    {
        try
        {
            int hoursToRun = 1;
            int flexibleTimeWindowMinutes = 10;

            var request = new CreateScheduleRequest
            {
                Name = name,

```

```

        ScheduleExpression = scheduleExpression,
        GroupName = scheduleGroupName,
        Target = new Target { Arn = targetArn, RoleArn = roleArn, Input =
input },

        ActionAfterCompletion = deleteAfterCompletion
            ? ActionAfterCompletion.DELETE
            : ActionAfterCompletion.NONE,
        StartDate = DateTime.UtcNow, // Ignored for one-time schedules.
        EndDate =
            DateTime.UtcNow
                .AddHours(hoursToRun) // Ignored for one-time schedules.
    };
    // Allow a flexible time window if the caller specifies it.
    request.FlexibleTimeWindow = new FlexibleTimeWindow
    {
        Mode = useFlexibleTimeWindow
            ? FlexibleTimeWindowMode.FLEXIBLE
            : FlexibleTimeWindowMode.OFF,
        MaximumWindowInMinutes = useFlexibleTimeWindow
            ? flexibleTimeWindowMinutes
            : null
    };

    var response = await _amazonScheduler.CreateScheduleAsync(request);

    Console.WriteLine($"Successfully created schedule '{name}' " +
        $"in schedule group '{scheduleGroupName}':
{response.ScheduleArn}.");
    return true;
}
catch (ConflictException ex)
{
    // If the name is not unique, a ConflictException will be thrown.
    _logger.LogError($"Failed to create schedule '{name}' due to a conflict.
{ex.Message}");
    return false;
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while creating schedule '{name}' "
+
        $"in schedule group '{scheduleGroupName}':
{ex.Message}");
    return false;
}

```

```
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateSchedule](#) à la section Référence des AWS SDK pour .NET API.

CreateScheduleGroup

L'exemple de code suivant montre comment utiliser `CreateScheduleGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Creates a new schedule group in Amazon EventBridge Scheduler.
/// </summary>
/// <param name="name">The name of the schedule group.</param>
/// <returns>True if the schedule group was created successfully, false
otherwise.</returns>
public async Task<bool> CreateScheduleGroupAsync(string name)
{
    try
    {
        var request = new CreateScheduleGroupRequest { Name = name };

        var response = await _amazonScheduler.CreateScheduleGroupAsync(request);

        Console.WriteLine($"Successfully created schedule group '{name}':
{response.ScheduleGroupArn}.");
        return true;
    }
    catch (ConflictException ex)
    {
        // If the name is not unique, a ConflictException will be thrown.
    }
}
```

```

        _logger.LogError($"Failed to create schedule group '{name}' due to a
conflict. {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while creating schedule group '{name}':
{ex.Message}");
        return false;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateScheduleGroup](#) à la section Référence des AWS SDK pour .NET API.

DeleteSchedule

L'exemple de code suivant montre comment utiliser `DeleteSchedule`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Deletes an existing schedule from Amazon EventBridge Scheduler.
/// </summary>
/// <param name="name">The name of the schedule to delete.</param>
/// <param name="groupName">The group name of the schedule to delete.</param>
/// <returns>True if the schedule was deleted successfully, false otherwise.</
returns>
public async Task<bool> DeleteScheduleAsync(string name, string groupName)
{
    try
    {
        var request = new DeleteScheduleRequest

```

```
        {
            Name = name,
            GroupName = groupName
        };

        await _amazonScheduler.DeleteScheduleAsync(request);

        Console.WriteLine($"Successfully deleted schedule with name '{name}'.");
        return true;

    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError(
            $"Failed to delete schedule with ID '{name}' because the resource
was not found: {ex.Message}");
        return true;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while deleting schedule with ID '{name}':
{ex.Message}");
        return false;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteSchedule](#) à la section Référence des AWS SDK pour .NET API.

DeleteScheduleGroup

L'exemple de code suivant montre comment utiliser `DeleteScheduleGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Deletes an existing schedule group from Amazon EventBridge Scheduler.
/// </summary>
/// <param name="name">The name of the schedule group to delete.</param>
/// <returns>True if the schedule group was deleted successfully, false
otherwise.</returns>
public async Task<bool> DeleteScheduleGroupAsync(string name)
{
    try
    {
        var request = new DeleteScheduleGroupRequest { Name = name };

        await _amazonScheduler.DeleteScheduleGroupAsync(request);

        Console.WriteLine($"Successfully deleted schedule group '{name}'.");
        return true;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError(
            $"Failed to delete schedule group '{name}' because the resource was
not found: {ex.Message}");
        return true;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while deleting schedule group '{name}':
{ex.Message}");
        return false;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteScheduleGroup](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Événements planifiés

L'exemple de code suivant illustre comment :

- Déployez une CloudFormation pile avec les ressources requises.
- Créez un groupe de planification EventBridge Scheduler.
- Créez un EventBridge calendrier ponctuel avec un créneau horaire flexible.
- Créez un calendrier de EventBridge planificateur récurrent avec un taux spécifié.
- Supprimez le EventBridge planificateur et le groupe de planification.
- nettoyer les ressources et supprimer la pile.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez le scénario.

```
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Scheduler;
using Amazon.Scheduler.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using SchedulerActions;
using Exception = System.Exception;

namespace SchedulerScenario;

public class SchedulerWorkflow
{

```



```

/*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET code example performs the following tasks for the Amazon EventBridge
    Scheduler workflow:

    1. Prepare the Application:
        - Prompt the user for an email address to use for the subscription for the
        SNS topic subscription.
        - Prompt the user for a name for the Cloud Formation stack.
        - Deploy the Cloud Formation template in resources/cfn_template.yaml for
        resource creation.
        - Store the outputs of the stack into variables for use in the scenario.
        - Create a schedule group for all schedules.

    2. Create one-time Schedule:
        - Create a one-time schedule to send an initial event.
        - Use a Flexible Time Window and set the schedule to delete after completion.
        - Wait for the user to receive the event email from SNS.

    3. Create a time-based schedule:
        - Prompt the user for how many X times per Y hours a recurring event should
        be scheduled.
        - Create the scheduled event for X times per hour for Y hours.
        - Wait for the user to receive the event email from SNS.
        - Delete the schedule when the user is finished.

    4. Clean up:
        - Prompt the user for y/n answer if they want to destroy the stack and clean
        up all resources.
        - Delete the schedule group.
        - Destroy the Cloud Formation stack and wait until the stack has been
        removed.
*/

public static ILogger<SchedulerWorkflow> _logger = null!;
public static SchedulerWrapper _schedulerWrapper = null!;
public static IAmazonCloudFormation _amazonCloudFormation = null!;

private static string _roleArn = null!;
private static string _snsTopicArn = null!;

public static bool _interactive = true;
private static string _stackName = "default-scheduler-scenario-stack-name";

```

```

private static string _scheduleGroupName = "scenario-schedules-group";
private static string _stackResourcePath = "../..../..../scenarios/
features/eventbridge_scheduler/resources/cfn_template.yaml";

public static async Task Main(string[] args)
{
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonScheduler>()
                .AddAWSService<IAmazonCloudFormation>()
                .AddTransient<SchedulerWrapper>()
            )
        .Build();

    if (!_interactive)
    {
        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<SchedulerWorkflow>();

        _schedulerWrapper =
host.Services.GetRequiredService<SchedulerWrapper>();
        _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
    }

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Amazon EventBridge Scheduler Scenario.");
    Console.WriteLine(new string('-', 80));

    try
    {
        Console.WriteLine(new string('-', 80));
        var prepareSuccess = await PrepareApplication();
        Console.WriteLine(new string('-', 80));

        if (prepareSuccess)
        {
            Console.WriteLine(new string('-', 80));
            await CreateOneTimeSchedule();
        }
    }
}

```

```

        Console.WriteLine(new string('-', 80));

        Console.WriteLine(new string('-', 80));
        await CreateRecurringSchedule();
        Console.WriteLine(new string('-', 80));
    }

    Console.WriteLine(new string('-', 80));
    await Cleanup();
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
    _interactive = false;
    await Cleanup();
}

Console.WriteLine("Amazon EventBridge Scheduler scenario completed.");
}

/// <summary>
/// Prepares the application by creating the necessary resources.
/// </summary>
/// <returns>True if the application was prepared successfully.</returns>
public static async Task<bool> PrepareApplication()
{
    Console.WriteLine("Preparing the application...");
    try
    {
        // Prompt the user for an email address to use for the subscription.
        Console.WriteLine("\nThis example creates resources in a CloudFormation
stack, including an SNS topic" +
            "\nthat will be subscribed to the EventBridge Scheduler
events. " +
            "\n\nYou will need to confirm the subscription in order to
receive event emails. ");

        var emailAddress = PromptUserForEmail();

        // Prompt the user for a name for the CloudFormation stack
        _stackName = PromptUserForStackName();
    }
}

```

```

        // Deploy the CloudFormation stack
        var deploySuccess = await DeployCloudFormationStack(_stackName,
emailAddress);

        if (deploySuccess)
        {
            // Create a schedule group for all schedules
            await
_schedulerWrapper.CreateScheduleGroupAsync(_scheduleGroupName);

            Console.WriteLine("Application preparation complete.");
            return true;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while preparing the
application.");
    }
    Console.WriteLine("Application preparation failed.");
    return false;
}

/// <summary>
/// Deploys the CloudFormation stack with the necessary resources.
/// </summary>
/// <param name="stackName">The name of the CloudFormation stack.</param>
/// <param name="email">The email to use for the subscription.</param>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName,
string email)
{
    Console.WriteLine($"\\nDeploying CloudFormation stack: {stackName}");

    try
    {
        var request = new CreateStackRequest
        {
            StackName = stackName,
            TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
            Capabilities = { Capability.CAPABILITY_NAMED_IAM }
        };

        // If an email is provided, set the parameter.

```

```
        if (!string.IsNullOrEmpty(email))
        {
            request.Parameters = new List<Parameter>()
            {
                new() { ParameterKey = "email", ParameterValue = email }
            };
        }

        var response = await _amazonCloudFormation.CreateStackAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

            // Wait for the stack to be in CREATE_COMPLETE state
            bool stackCreated = await WaitForStackCompletion(response.StackId);

            if (stackCreated)
            {
                // Retrieve the output values
                var success = await GetStackOutputs(response.StackId);
                return success;
            }
            else
            {
                _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                return false;
            }
        }
        else
        {
            _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
            return false;
        }
    }
    catch (AlreadyExistsException)
    {
        _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
        var newStackName = PromptUserForStackName();
        return await DeployCloudFormationStack(newStackName, email);
    }
}
```

```

    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
        return false;
    }
}

/// <summary>
/// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
/// </summary>
/// <param name="client">The CloudFormation client.</param>
/// <param name="stackId">The ID of the CloudFormation stack.</param>
/// <returns>True if the stack was created successfully.</returns>
private static async Task<bool> WaitForStackCompletion(string stackId)
{
    int retryCount = 0;
    const int maxRetries = 10;
    const int retryDelay = 30000; // 30 seconds.

    while (retryCount < maxRetries)
    {
        var describeStacksRequest = new DescribeStacksRequest
        {
            StackName = stackId
        };

        var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

        if (describeStacksResponse.Stacks.Count > 0)
        {
            if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
            {
                Console.WriteLine("CloudFormation stack creation complete.");
                return true;
            }
            if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
            {

```

```

        Console.WriteLine("CloudFormation stack creation failed.");
        return false;
    }
}

    Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
    await Task.Delay(retryDelay);
    retryCount++;
}

    _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
    return false;
}

/// <summary>
/// Retrieves the output values from the CloudFormation stack.
/// </summary>
/// <param name="stackId">The ID of the CloudFormation stack.</param>
private static async Task<bool> GetStackOutputs(string stackId)
{
    try
    {
        var describeStacksRequest = new DescribeStacksRequest { StackName =
stackId };

        var describeStacksResponse =
            await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

        if (describeStacksResponse.Stacks.Count > 0)
        {
            var stack = describeStacksResponse.Stacks[0];
            _roleArn = GetStackOutputValue(stack, "RoleARN");
            _snsTopicArn = GetStackOutputValue(stack, "SNSStopicARN");
            return true;
        }
        else
        {
            _logger.LogError($"No stack found for stack outputs: {stackId}");
            return false;
        }
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError(
                ex, $"Failed to retrieve CloudFormation stack outputs: {stackId}");
            return false;
        }
    }

    /// <summary>
    /// Get an output value by key from a CloudFormation stack.
    /// </summary>
    /// <param name="stack">The CloudFormation stack.</param>
    /// <param name="outputKey">The key of the output.</param>
    /// <returns>The value as a string.</returns>
    private static string GetStackOutputValue(Stack stack, string outputKey)
    {
        var output = stack.Outputs.First(o => o.OutputKey == outputKey);
        var outputValue = output.OutputValue;
        Console.WriteLine($"Stack output {outputKey}: {outputValue}");
        return outputValue;
    }

    /// <summary>
    /// Creates a one-time schedule to send an initial event.
    /// </summary>
    /// <returns>True if the one-time schedule was created successfully.</returns>
    public static async Task<bool> CreateOneTimeSchedule()
    {
        var scheduleName =
            PromptUserForResourceName("Enter a name for the one-time schedule:");

        Console.WriteLine($"Creating a one-time schedule named '{scheduleName}' " +
            $"to send an initial event in 1 minute with a flexible
time window...");
        try
        {
            // Create a one-time schedule with a flexible time
            // window set to delete after completion.
            // You may also set a timezone instead of using UTC.
            var scheduledTime = DateTime.UtcNow.AddMinutes(1).ToString("s");

            var createSuccess = await _schedulerWrapper.CreateScheduleAsync(
                scheduleName,
                $"at({scheduledTime})",

```



```

        _scheduleGroupName,
        _snsTopicArn,
        _roleArn,
        $"One time scheduled event test from schedule {scheduleName}.",
        true,
        useFlexibleTimeWindow: true);

    Console.WriteLine($"Subscription email will receive an email from this
event.");
    Console.WriteLine($"You must confirm your subscription to receive event
emails.");

    Console.WriteLine($"One-time schedule '{scheduleName}' created
successfully.");
    return createSuccess;
}
catch (ResourceNotFoundException ex)
{
    _logger.LogError(ex, $"The target with ARN '{_snsTopicArn}' was not
found.");
    return false;
}
catch (Exception ex)
{
    _logger.LogError(ex, $"An error occurred while creating the one-time
schedule '{scheduleName}'.");
    return false;
}
}

/// <summary>
/// Create a recurring schedule to send events at a specified rate in minutes.
/// </summary>
/// <returns>True if the recurring schedule was created successfully.</returns>
public static async Task<bool> CreateRecurringSchedule()
{
    Console.WriteLine("Creating a recurring schedule to send events for one
hour...");

    try
    {
        // Prompt the user for a schedule name.
        var scheduleName =

```

```

        PromptUserForResourceName("Enter a name for the recurring schedule:
");

        // Prompt the user for the schedule rate (in minutes).
        var scheduleRateInMinutes =
            PromptUserForInteger("Enter the desired schedule rate (in minutes):
");

        // Create the recurring schedule.
        var createSuccess = await _schedulerWrapper.CreateScheduleAsync(
            scheduleName,
            $"rate({scheduleRateInMinutes} minutes)",
            _scheduleGroupName,
            _snsTopicArn,
            _roleArn,
            $"Recurrent event test from schedule {scheduleName}.");

        Console.WriteLine($"Subscription email will receive an email from this
event.");
        Console.WriteLine($"You must confirm your subscription to receive event
emails.");

        // Delete the schedule when the user is finished.
        if (!_interactive || GetYesNoResponse($"Are you ready to delete the
'{scheduleName}' schedule? (y/n)"))
        {
            await _schedulerWrapper.DeleteScheduleAsync(scheduleName,
            _scheduleGroupName);
        }

        return createSuccess;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError(ex, $"The target with ARN '{_snsTopicArn}' was not
found.");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while creating the recurring
schedule.");
        return false;
    }
}

```

```

    }

    /// <summary>
    /// Cleans up the resources created during the scenario.
    /// </summary>
    /// <returns>True if the cleanup was successful.</returns>
    public static async Task<bool> Cleanup()
    {
        // Prompt the user to confirm cleanup.
        var cleanup = !_interactive || GetYesNoResponse(
            "Do you want to delete all resources created by this scenario? (y/n) ");
        if (cleanup)
        {
            try
            {
                // Delete the schedule group.
                var groupDeleteSuccess = await
                _schedulerWrapper.DeleteScheduleGroupAsync(_scheduleGroupName);

                // Destroy the CloudFormation stack and wait for it to be removed.
                var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
                false);

                return groupDeleteSuccess && stackDeleteSuccess;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex,
                    "An error occurred while cleaning up the resources.");
                return false;
            }
        }
        _logger.LogInformation("EventBridge Scheduler scenario is complete.");
        return true;
    }

    /// <summary>
    /// Delete the resources in the stack and wait for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the stack.</param>
    /// <param name="forceDelete">True to force delete the stack.</param>
    /// <returns>True if successful.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
    forceDelete)

```

```

{
    var request = new DeleteStackRequest
    {
        StackName = stackName,
    };

    if (forceDelete)
    {
        request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
    }

    await _amazonCloudFormation.DeleteStackAsync(request);
    Console.WriteLine($"CloudFormation stack '{_stackName}' is being deleted.
This may take a few minutes.");

    bool stackDeleted = await WaitForStackDeletion(_stackName, forceDelete);

    if (stackDeleted)
    {
        Console.WriteLine($"CloudFormation stack '{_stackName}' has been
deleted.");
        return true;
    }
    else
    {
        _logger.LogError($"Failed to delete CloudFormation stack
'{_stackName}'.");
        return false;
    }
}

/// <summary>
/// Wait for the stack to be deleted.
/// </summary>
/// <param name="stackName">The name of the stack.</param>
/// <param name="forceDelete">True to force delete the stack.</param>
/// <returns>True if successful.</returns>
private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
{
    int retryCount = 0;
    const int maxRetries = 10;
    const int retryDelay = 30000; // 30 seconds

```

```

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
                _amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                describeStacksResponse.Stacks[0].StackStatus == StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
                StackStatus.DELETE_FAILED)
                {
                    // Try one time to force delete.
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
            "ValidationError")
            {
                // Stack does not exist, so it has been successfully deleted.
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
            deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
        to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.

```

```

    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompt the user for a valid email address.
    /// </summary>
    /// <returns>The valid email address.</returns>
    private static string PromptUserForEmail()
    {
        if (!_interactive)
        {
            Console.WriteLine("Enter an email address to use for event
subscriptions: ");

            string email = Console.ReadLine!();

            if (!IsValidEmail(email))
            {
                Console.WriteLine("Invalid email address. Please try again.");
                return PromptUserForEmail();
            }
            return email;
        }
        // Used when running without user prompts.
        return "";
    }

    /// <summary>
    /// Prompt the user for a non-empty stack name.
    /// </summary>
    /// <returns>The valid stack name</returns>
    private static string PromptUserForStackName()
    {
        Console.WriteLine("Enter a name for the AWS Cloud Formation Stack: ");
        if (!_interactive)

```

```

    {
        string stackName = Console.ReadLine()!;
        var regex = "[a-zA-Z][-a-zA-Z0-9]|arn:[-a-zA-Z0-9:/._+]";
        if (!Regex.IsMatch(stackName, regex))
        {
            Console.WriteLine(
                $"Invalid stack name. Please use a name that matches the pattern
{regex}.");
            return PromptUserForStackName();
        }

        return stackName;
    }
    // Used when running without user prompts.
    return _stackName;
}

/// <summary>
/// Prompt the user for a non-empty resource name.
/// </summary>
/// <returns>The valid stack name</returns>
private static string PromptUserForResourceName(string prompt)
{
    if (_interactive)
    {
        Console.WriteLine(prompt);
        string resourceName = Console.ReadLine()!;
        var regex = "[0-9a-zA-Z-_.]+";
        if (!Regex.IsMatch(resourceName, regex))
        {
            Console.WriteLine($"Invalid resource name. Please use a name that
matches the pattern {regex}.");
            return PromptUserForResourceName(prompt);
        }
        return resourceName!;
    }
    // Used when running without user prompts.
    return "resource-" + Guid.NewGuid();
}

/// <summary>
/// Prompt the user for a non-empty resource name.
/// </summary>
/// <returns>The valid stack name</returns>

```

```

private static int PromptUserForInteger(string prompt)
{
    if (_interactive)
    {
        Console.WriteLine(prompt);
        string stringResponse = Console.ReadLine();
        if (string.IsNullOrEmpty(stringResponse) ||
            !Int32.TryParse(stringResponse, out var intResponse))
        {
            Console.WriteLine($"Invalid integer. ");
            return PromptUserForInteger(prompt);
        }
        return intResponse!;
    }
    // Used when running without user prompts.
    return 1;
}

/// <summary>
/// Use System Mail to check for a valid email address.
/// </summary>
/// <param name="email">The string to verify.</param>
/// <returns>True if a valid email address.</returns>
private static bool IsValidEmail(string email)
{
    try
    {
        var mailAddress = new System.Net.Mail.MailAddress(email);
        return mailAddress.Address == email;
    }
    catch
    {
        // Invalid emails will cause an exception, return false.
        return false;
    }
}
}

```

Encapsuleur pour les opérations de service.

```

using Amazon.Scheduler;
using Amazon.Scheduler.Model;

```



```

using Microsoft.Extensions.Logging;

namespace SchedulerActions;

/// <summary>
/// Wrapper class for Amazon EventBridge Scheduler operations.
/// </summary>
public class SchedulerWrapper
{
    private readonly IAmazonScheduler _amazonScheduler;
    private readonly ILogger<SchedulerWrapper> _logger;

    /// <summary>
    /// Constructor for the SchedulerWrapper class.
    /// </summary>
    /// <param name="amazonScheduler">The injected EventBridge Scheduler client.</
param>
    /// <param name="logger">The injected logger.</param>
    public SchedulerWrapper(IAmazonScheduler amazonScheduler,
        ILogger<SchedulerWrapper> logger)
    {
        _amazonScheduler = amazonScheduler;
        _logger = logger;
    }

    /// <summary>
    /// Creates a new schedule in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule.</param>
    /// <param name="scheduleExpression">The schedule expression that defines when
the schedule should run.</param>
    /// <param name="scheduleGroupName">The name of the schedule group to which the
schedule should be added.</param>
    /// <param name="deleteAfterCompletion">Indicates whether to delete the schedule
after completion.</param>
    /// <param name="useFlexibleTimeWindow">Indicates whether to use a flexible time
window for the schedule.</param>
    /// <param name="targetArn">ARN of the event target.</param>
    /// <param name="roleArn">Execution Role ARN.</param>
    /// <returns>True if the schedule was created successfully, false otherwise.</
returns>
    public async Task<bool> CreateScheduleAsync(
        string name,
        string scheduleExpression,

```

```

        string scheduleGroupName,
        string targetArn,
        string roleArn,
        string input,
        bool deleteAfterCompletion = false,
        bool useFlexibleTimeWindow = false)
    {
        try
        {
            int hoursToRun = 1;
            int flexibleTimeWindowMinutes = 10;

            var request = new CreateScheduleRequest
            {
                Name = name,
                ScheduleExpression = scheduleExpression,
                GroupName = scheduleGroupName,
                Target = new Target { Arn = targetArn, RoleArn = roleArn, Input =
input },
                ActionAfterCompletion = deleteAfterCompletion
                    ? ActionAfterCompletion.DELETE
                    : ActionAfterCompletion.NONE,
                StartDate = DateTime.UtcNow, // Ignored for one-time schedules.
                EndDate =
                    DateTime.UtcNow
                        .AddHours(hoursToRun) // Ignored for one-time schedules.
            };
            // Allow a flexible time window if the caller specifies it.
            request.FlexibleTimeWindow = new FlexibleTimeWindow
            {
                Mode = useFlexibleTimeWindow
                    ? FlexibleTimeWindowMode.FLEXIBLE
                    : FlexibleTimeWindowMode.OFF,
                MaximumWindowInMinutes = useFlexibleTimeWindow
                    ? flexibleTimeWindowMinutes
                    : null
            };

            var response = await _amazonScheduler.CreateScheduleAsync(request);

            Console.WriteLine($"Successfully created schedule '{name}' " +
                             $"in schedule group '{scheduleGroupName}':
{response.ScheduleArn}.");
            return true;
        }
    }
}

```

```

    }
    catch (ConflictException ex)
    {
        // If the name is not unique, a ConflictException will be thrown.
        _logger.LogError($"Failed to create schedule '{name}' due to a conflict.
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while creating schedule '{name}' "
+
                                $"in schedule group '{scheduleGroupName}':
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Creates a new schedule group in Amazon EventBridge Scheduler.
/// </summary>
/// <param name="name">The name of the schedule group.</param>
/// <returns>True if the schedule group was created successfully, false
otherwise.</returns>
public async Task<bool> CreateScheduleGroupAsync(string name)
{
    try
    {
        var request = new CreateScheduleGroupRequest { Name = name };

        var response = await _amazonScheduler.CreateScheduleGroupAsync(request);

        Console.WriteLine($"Successfully created schedule group '{name}':
{response.ScheduleGroupArn}.");
        return true;
    }
    catch (ConflictException ex)
    {
        // If the name is not unique, a ConflictException will be thrown.
        _logger.LogError($"Failed to create schedule group '{name}' due to a
conflict. {ex.Message}");
        return false;
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while creating schedule group '{name}':
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an existing schedule from Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule to delete.</param>
    /// <param name="groupName">The group name of the schedule to delete.</param>
    /// <returns>True if the schedule was deleted successfully, false otherwise.</
returns>
    public async Task<bool> DeleteScheduleAsync(string name, string groupName)
    {
        try
        {
            var request = new DeleteScheduleRequest
            {
                Name = name,
                GroupName = groupName
            };

            await _amazonScheduler.DeleteScheduleAsync(request);

            Console.WriteLine($"Successfully deleted schedule with name '{name}'.");
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError(
                $"Failed to delete schedule with ID '{name}' because the resource
was not found: {ex.Message}");
            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while deleting schedule with ID '{name}':
{ex.Message}");

```

```

        return false;
    }
}

/// <summary>
/// Deletes an existing schedule group from Amazon EventBridge Scheduler.
/// </summary>
/// <param name="name">The name of the schedule group to delete.</param>
/// <returns>True if the schedule group was deleted successfully, false
otherwise.</returns>
public async Task<bool> DeleteScheduleGroupAsync(string name)
{
    try
    {
        var request = new DeleteScheduleGroupRequest { Name = name };

        await _amazonScheduler.DeleteScheduleGroupAsync(request);

        Console.WriteLine($"Successfully deleted schedule group '{name}'.");
        return true;
    }
    catch (ResourceNotFoundException ex)
    {
        _logger.LogError(
            $"Failed to delete schedule group '{name}' because the resource was
not found: {ex.Message}");
        return true;
    }
    catch (Exception ex)
    {
        _logger.LogError(
            $"An error occurred while deleting schedule group '{name}':
{ex.Message}");
        return false;
    }
}
}

```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateSchedule](#)

- [CreateScheduleGroup](#)
- [DeleteSchedule](#)
- [DeleteScheduleGroups](#)

Exemples d'utilisation d'Amazon Glacier SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Glacier.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Actions](#)

Mise en route

Bonjour Amazon Glacier

L'exemple de code suivant montre comment commencer à utiliser Amazon Glacier.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.Glacier;  
using Amazon.Glacier.Model;
```

```
namespace GlacierActions;

public static class HelloGlacier
{
    static async Task Main()
    {
        var glacierService = new AmazonGlacierClient();

        Console.WriteLine("Hello Amazon Glacier!");
        Console.WriteLine("Let's list your Glacier vaults:");

        // You can use await and any of the async methods to get a response.
        // Let's get the vaults using a paginator.
        var glacierVaultPaginator = glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            Console.WriteLine($"{vault.CreationDate}:{vault.VaultName}, ARN:
{vault.VaultARN}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListVaults](#) à la section Référence des AWS SDK pour .NET API.

Actions

AddTagsToVault

L'exemple de code suivant montre comment utiliser `AddTagsToVault`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Add tags to the items in an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to add tags to.</param>
    /// <param name="key">The name of the object to tag.</param>
    /// <param name="value">The tag value to add.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddTagsToVaultAsync(string vaultName, string key, string
value)
    {
        var request = new AddTagsToVaultRequest
        {
            Tags = new Dictionary<string, string>
            {
                { key, value },
            },
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.AddTagsToVaultAsync(request);
        return response.HttpStatusCode == HttpStatusCode.NoContent;
    }

```

- Pour plus de détails sur l'API, reportez-vous [AddTagsToVault](#) à la section Référence des AWS SDK pour .NET API.

CreateVault

L'exemple de code suivant montre comment utiliser `CreateVault`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>

```



```

/// Create an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to create.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateVaultAsync(string vaultName)
{
    var request = new CreateVaultRequest
    {
        // Setting the AccountId to "-" means that
        // the account associated with the current
        // account will be used.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.CreateVaultAsync(request);

    Console.WriteLine($"Created {vaultName} at: {response.Location}");

    return response.HttpStatusCode == HttpStatusCode.Created;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateVault](#) à la section Référence des AWS SDK pour .NET API.

DescribeVault

L'exemple de code suivant montre comment utiliser `DescribeVault`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Describe an Amazon S3 Glacier vault.
/// </summary>

```

```
/// <param name="vaultName">The name of the vault to describe.</param>
/// <returns>The Amazon Resource Name (ARN) of the vault.</returns>
public async Task<string> DescribeVaultAsync(string vaultName)
{
    var request = new DescribeVaultRequest
    {
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.DescribeVaultAsync(request);

    // Display the information about the vault.
    Console.WriteLine($"{response.VaultName}\tARN: {response.VaultARN}");
    Console.WriteLine($"Created on: {response.CreationDate}\tNumber of Archives:
{response.NumberOfArchives}\tSize (in bytes): {response.SizeInBytes}");
    if (response.LastInventoryDate != DateTime.MinValue)
    {
        Console.WriteLine($"Last inventory: {response.LastInventoryDate}");
    }

    return response.VaultARN;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeVault](#) à la section Référence des AWS SDK pour .NET API.

InitiateJob

L'exemple de code suivant montre comment utiliser `InitiateJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Extrayez une archive d'un coffre-fort. Cet exemple utilise la `ArchiveTransferManager` classe. Pour plus de détails sur l'API, voir [ArchiveTransferManager](#).

```

    /// <summary>
    /// Download an archive from an Amazon S3 Glacier vault using the Archive
    /// Transfer Manager.
    /// </summary>
    /// <param name="vaultName">The name of the vault containing the object.</param>
    /// <param name="archiveId">The Id of the archive to download.</param>
    /// <param name="localFilePath">The local directory where the file will
    /// be stored after download.</param>
    /// <returns>Async Task.</returns>
    public async Task<bool> DownloadArchiveWithArchiveManagerAsync(string vaultName,
string archiveId, string localFilePath)
    {
        try
        {
            var manager = new ArchiveTransferManager(_glacierService);

            var options = new DownloadOptions
            {
                StreamTransferProgress = Progress!,
            };

            // Download an archive.
            Console.WriteLine("Initiating the archive retrieval job and then polling
SQS queue for the archive to be available.");
            Console.WriteLine("When the archive is available, downloading will
begin.");
            await manager.DownloadAsync(vaultName, archiveId, localFilePath,
options);

            return true;
        }
        catch (AmazonGlacierException ex)
        {
            Console.WriteLine(ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Event handler to track the progress of the Archive Transfer Manager.

```

```

/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="args">The argument values from the object that raised the
/// event.</param>
static void Progress(object sender, StreamTransferProgressArgs args)
{
    if (args.PercentDone != _currentPercentage)
    {
        _currentPercentage = args.PercentDone;
        Console.WriteLine($"Downloaded {_currentPercentage}%");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [InitiateJob](#) à la section Référence des AWS SDK pour .NET API.

ListJobs

L'exemple de code suivant montre comment utiliser `ListJobs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// List Amazon S3 Glacier jobs.
/// </summary>
/// <param name="vaultName">The name of the vault to list jobs for.</param>
/// <returns>A list of Amazon S3 Glacier jobs.</returns>
public async Task<List<GlacierJobDescription>> ListJobsAsync(string vaultName)
{
    var request = new ListJobsRequest
    {
        // Using a hyphen "-" for the Account Id will
        // cause the SDK to use the Account Id associated
    }
}

```

```

        // with the current account.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.ListJobsAsync(request);

    return response.JobList;
}

```

- Pour plus de détails sur l'API, reportez-vous [ListJobs](#) à la section Référence des AWS SDK pour .NET API.

ListTagsForVault

L'exemple de code suivant montre comment utiliser `ListTagsForVault`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// List tags for an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to list tags for.</param>
/// <returns>A dictionary listing the tags attached to each object in the
/// vault and its tags.</returns>
public async Task<Dictionary<string, string>> ListTagsForVaultAsync(string
vaultName)
{
    var request = new ListTagsForVaultRequest
    {
        // Using a hyphen "-" for the Account Id will
        // cause the SDK to use the Account Id associated
        // with the default user.
    }
}

```

```

        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.ListTagsForVaultAsync(request);

    return response.Tags;
}

```

- Pour plus de détails sur l'API, reportez-vous [ListTagsForVault](#) à la section Référence des AWS SDK pour .NET API.

ListVaults

L'exemple de code suivant montre comment utiliser `ListVaults`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// List the Amazon S3 Glacier vaults associated with the current account.
/// </summary>
/// <returns>A list containing information about each vault.</returns>
public async Task<List<DescribeVaultOutput>> ListVaultsAsync()
{
    var glacierVaultPaginator = _glacierService.Paginators.ListVaults(
        new ListVaultsRequest { AccountId = "-" });
    var vaultList = new List<DescribeVaultOutput>();

    await foreach (var vault in glacierVaultPaginator.VaultList)
    {
        vaultList.Add(vault);
    }
}

```

```
        return vaultList;
    }
```

- Pour plus de détails sur l'API, reportez-vous [ListVaults](#) à la section Référence des AWS SDK pour .NET API.

UploadArchive

L'exemple de code suivant montre comment utiliser `UploadArchive`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Upload an object to an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the Amazon S3 Glacier vault to upload
/// the archive to.</param>
/// <param name="archiveFilePath">The file path of the archive to upload to the
vault.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<string> UploadArchiveWithArchiveManager(string vaultName,
string archiveFilePath)
{
    try
    {
        var manager = new ArchiveTransferManager(_glacierService);

        // Upload an archive.
        var response = await manager.UploadAsync(vaultName, "upload archive
test", archiveFilePath);
        return response.ArchiveId;
    }
    catch (AmazonGlacierException ex)
```

```
{  
    Console.WriteLine(ex.Message);  
    return string.Empty;  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [UploadArchive](#) à la section Référence des AWS SDK pour .NET API.

AWS Glue exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with AWS Glue.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour AWS Glue

L'exemple de code suivant montre comment démarrer avec AWS Glue.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                    .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
```

```
    }
    while (request.NextToken is not null);

    Console.Clear();
    Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
    if (jobNames.Count == 0)
    {
        Console.WriteLine("You don't have any AWS Glue jobs.");
    }
    else
    {
        jobNames.ForEach(Console.WriteLine);
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListJobs](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un Crawler qui indexe un compartiment Amazon S3 public et génère une base de données de métadonnées au format CSV.
- Répertoriez les informations relatives aux bases de données et aux tables de votre AWS Glue Data Catalog.
- Créez une tâche pour extraire les données CSV du compartiment S3, transformer les données et charger la sortie au format JSON dans un autre compartiment S3.
- Répertoriez les informations relatives aux exécutions de tâches, visualisez les données transformées et nettoyez les ressources.

Pour plus d'informations, consultez [Tutoriel : prise en main de AWS Glue Studio](#).

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une classe qui englobe les AWS Glue fonctions utilisées dans le scénario.

```
using System.Net;

namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
public async Task<bool> CreateCrawlerAsync(
    string crawlerName,
    string crawlerDescription,
    string role,
    string schedule,
    string s3Path,
    string dbName)
{
    var s3Target = new S3Target
    {
        Path = s3Path,
    };

    var targetList = new List<S3Target>
    {
        s3Target,
    };

    var targets = new CrawlerTargets
    {
        S3Targets = targetList,
    };

    var crawlerRequest = new CreateCrawlerRequest
    {
        DatabaseName = dbName,
        Name = crawlerName,
        Description = crawlerDescription,
        Targets = targets,
        Role = role,
        Schedule = schedule,
    };

    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Create an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="roleName">The name of the IAM role to be assumed by
/// the job.</param>
```

```

    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {

```

```

        var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

```

```

    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }

    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }

    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>

```

```

    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }

    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }

    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us behind
the scenes

```



```

        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }

    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }

    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();
    }

```

```
        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }

    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
        string inputTable,
        string bucketName)
    {
        var request = new StartJobRunRequest
        {
            JobName = jobName,
```

```

        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
}

```

Créez une classe qui exécute le scénario.

```

global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

using Amazon.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
    }
}

```

```

using var host = Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
        logging.AddFilter("System", LogLevel.Debug)
            .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
            .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonGlue>()
        .AddTransient<GlueWrapper>()
        .AddTransient<UiWrapper>()
        )
    .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<GlueBasics>();

_configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load settings from .json file.
    .AddJsonFile("settings.local.json",
        true) // Optionally load local settings.
    .Build();

// These values are stored in settings.json
// Once you have run the CDK script to deploy the resources,
// edit the file to set "BucketName", "RoleName", and "ScriptURL"
// to the appropriate values. Also set "CrawlerName" to the name
// you want to give the crawler when it is created.
string bucketName = _configuration["BucketName"]!;
string bucketUrl = _configuration["BucketUrl"]!;
string crawlerName = _configuration["CrawlerName"]!;
string roleName = _configuration["RoleName"]!;
string sourceData = _configuration["SourceData"]!;
string dbName = _configuration["DbName"]!;
string cron = _configuration["Cron"]!;
string scriptUrl = _configuration["ScriptURL"]!;
string jobName = _configuration["JobName"]!;

var wrapper = host.Services.GetRequiredService<GlueWrapper>();
var uiWrapper = host.Services.GetRequiredService<UiWrapper>();

uiWrapper.DisplayOverview();
uiWrapper.PressEnter();

```

```
// Create the crawler and wait for it to be ready.
uiWrapper.DisplayTitle("Create AWS Glue crawler");
Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
if (crawlerCreated)
{
    Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
    CrawlerState crawlerState;
    do
    {
        crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
    }
    while (crawlerState != "READY");
    Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
}
else
{
    Console.WriteLine($"Couldn't create crawler {crawlerName}.");
    return; // Exit the application.
}

uiWrapper.DisplayTitle("Start AWS Glue crawler");
Console.WriteLine("Now let's wait until the crawler has successfully
started.");
var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
if (crawlerStarted)
{
    CrawlerState crawlerState;
    do
    {
        crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
    }
    while (crawlerState != "READY");
    Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
}
else
{
    Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
    return; // Exit the application.
}
```

```
}

uiWrapper.PressEnter();

Console.WriteLine($"\\nLet's take a look at the database: {dbName}");
var database = await wrapper.GetDatabaseAsync(dbName);

if (database != null)
{
    uiWrapper.DisplayTitle($"{database.Name} Details");
    Console.WriteLine($"{database.Name} created on {database.CreateTime}");
    Console.WriteLine(database.Description);
}

uiWrapper.PressEnter();

var tables = await wrapper.GetTablesAsync(dbName);
if (tables.Count > 0)
{
    tables.ForEach(table =>
    {
        Console.WriteLine($"{table.Name}\\tCreated:
{table.CreateTime}\\tUpdated: {table.UpdateTime}");
    });
}

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Create AWS Glue job");
Console.WriteLine("Creating a new AWS Glue job.");
var description = "An AWS Glue job created using the AWS SDK for .NET";
await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Starting AWS Glue job");
Console.WriteLine("Starting the new AWS Glue job...");
var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
var jobRunComplete = false;
var jobRun = new JobRun();
do
{
```

```
        jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
        if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState == "STOPPED"
||
            jobRun.JobRunState == "FAILED" || jobRun.JobRunState == "TIMEOUT")
        {
            jobRunComplete = true;
        }
    } while (!jobRunComplete);

    uiWrapper.DisplayTitle($"Data in {bucketName}");

    // Get the list of data stored in the S3 bucket.
    var s3Client = new AmazonS3Client();

    var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
    response.S3Objects.ForEach(s3Object =>
    {
        Console.WriteLine(s3Object.Key);
    });

    uiWrapper.DisplayTitle("AWS Glue jobs");
    var jobNames = await wrapper.ListJobsAsync();
    jobNames.ForEach(jobName =>
    {
        Console.WriteLine(jobName);
    });

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Get AWS Glue job run information");
    Console.WriteLine("Getting information about the AWS Glue job.");
    var jobRuns = await wrapper.GetJobRunsAsync(jobName);

    jobRuns.ForEach(jobRun =>
    {
        Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
    });

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Deleting resources");
    Console.WriteLine("Deleting the AWS Glue job used by the example.");
```

```

        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });

        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);

        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}

namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the URL
to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
    }
}

```



```

        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }
}

```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Actions

CreateCrawler

L'exemple de code suivant montre comment utiliser `CreateCrawler`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
```

```

    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,

```

```

        Schedule = schedule,
    };

    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateCrawler](#) à la section Référence des AWS SDK pour .NET API.

CreateJob

L'exemple de code suivant montre comment utiliser `CreateJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="roleName">The name of the IAM role to be assumed by
/// the job.</param>
/// <param name="description">A description of the job.</param>
/// <param name="scriptUrl">The URL to the script.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
{
    var command = new JobCommand
    {
        PythonVersion = "3",
        Name = "glueetl",
        ScriptLocation = scriptUrl,
    }
}

```

```
};

var arguments = new Dictionary<string, string>
{
    { "--input_database", dbName },
    { "--input_table", tableName },
    { "--output_bucket_url", bucketUrl }
};

var request = new CreateJobRequest
{
    Command = command,
    DefaultArguments = arguments,
    Description = description,
    GlueVersion = "3.0",
    Name = jobName,
    NumberOfWorkers = 10,
    Role = roleName,
    WorkerType = "G.1X"
};

var response = await _amazonGlue.CreateJobAsync(request);
return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateJob](#) à la section Référence des AWS SDK pour .NET API.

DeleteCrawler

L'exemple de code suivant montre comment utiliser `DeleteCrawler`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteCrawlerAsync(string crawlerName)
{
    var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
    { Name = crawlerName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteCrawler](#) à la section Référence des AWS SDK pour .NET API.

DeleteDatabase

L'exemple de code suivant montre comment utiliser `DeleteDatabase`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete the AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteDatabaseAsync(string dbName)
{
    var response = await _amazonGlue.DeleteDatabaseAsync(new
    DeleteDatabaseRequest { Name = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteDatabase](#) à la section Référence des AWS SDK pour .NET API.

DeleteJob

L'exemple de code suivant montre comment utiliser `DeleteJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteJobAsync(string jobName)
{
    var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
    { JobName = jobName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteJob](#) à la section Référence des AWS SDK pour .NET API.

DeleteTable

L'exemple de code suivant montre comment utiliser `DeleteTable`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a table from an AWS Glue database.
/// </summary>
/// <param name="tableName">The table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTableAsync(string dbName, string tableName)
{
    var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
    { Name = tableName, DatabaseName = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteTable](#) à la section Référence des AWS SDK pour .NET API.

GetCrawler

L'exemple de code suivant montre comment utiliser `GetCrawler`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information about an AWS Glue crawler.
```



```

/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Crawler object describing the crawler.</returns>
public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new GetCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var databaseName = response.Crawler.DatabaseName;
        Console.WriteLine($"{crawlerName} has the database {databaseName}");
        return response.Crawler;
    }

    Console.WriteLine($"No information regarding {crawlerName} could be
found.");
    return null;
}

```

- Pour plus de détails sur l'API, reportez-vous [GetCrawler](#) à la section Référence des AWS SDK pour .NET API.

GetDatabase

L'exemple de code suivant montre comment utiliser `GetDatabase`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>

```

```
/// Get information about an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Database object containing information about the database.</
returns>
public async Task<Database> GetDatabaseAsync(string dbName)
{
    var databasesRequest = new GetDatabaseRequest
    {
        Name = dbName,
    };

    var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
    return response.Database;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetDatabase](#) à la section Référence des AWS SDK pour .NET API.

GetJobRun

L'exemple de code suivant montre comment utiliser `GetJobRun`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information about a specific AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="jobRunId">The Id of the job run.</param>
/// <returns>A JobRun object with information about the job run.</returns>
public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
{
```

```

        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
        { JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }

```

- Pour plus de détails sur l'API, reportez-vous [GetJobRun](#) à la section Référence des AWS SDK pour .NET API.

GetJobRuns

L'exemple de code suivant montre comment utiliser `GetJobRuns`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get information about all AWS Glue runs of a specific job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A list of JobRun objects.</returns>
public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };

    // No need to loop to get all the log groups--the SDK does it for us behind
    the scenes
    var paginatorForJobRuns =
        _amazonGlue.Paginators.GetJobRuns(request);

```

```

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }

```

- Pour plus de détails sur l'API, reportez-vous [GetJobRuns](#) à la section Référence des AWS SDK pour .NET API.

GetTables

L'exemple de code suivant montre comment utiliser `GetTables`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

```

```
await foreach (var response in tablePaginator.Responses)
{
    tables.AddRange(response.TableList);
}

return tables;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetTables](#) à la section Référence des AWS SDK pour .NET API.

ListJobs

L'exemple de code suivant montre comment utiliser `ListJobs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List AWS Glue jobs using a paginator.
/// </summary>
/// <returns>A list of AWS Glue job names.</returns>
public async Task<List<string>> ListJobsAsync()
{
    var jobNames = new List<string>();

    var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
    { MaxResults = 10 });
    await foreach (var response in listJobsPaginator.Responses)
    {
        jobNames.AddRange(response.JobNames);
    }

    return jobNames;
}
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [ListJobs](#) à la section Référence des AWS SDK pour .NET API.

StartCrawler

L'exemple de code suivant montre comment utiliser `StartCrawler`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [StartCrawler](#) à la section Référence des AWS SDK pour .NET API.

StartJobRun

L'exemple de code suivant montre comment utiliser `StartJobRun`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- Pour plus de détails sur l'API, reportez-vous [StartJobRun](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'IAM utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with IAM.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)
- [Scénarios](#)

Mise en route

Bonjour IAM

L'exemple de code suivant montre comment commencer à utiliser IAM.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).


```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();

        var listPoliciesPaginator = iamClient.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
{policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListPolicies](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant montre comment créer un utilisateur et endosser un rôle.

⚠ Warning

Afin d'éviter les risques de sécurité, n'employez pas les utilisateurs IAM pour l'authentification lorsque vous développez des logiciels spécialisés ou lorsque vous travaillez avec des données réelles. Préférez la fédération avec un fournisseur d'identité tel que [AWS IAM Identity Center](#).

- Créer un utilisateur sans autorisation.
- Créer un rôle qui accorde l'autorisation de répertorier les compartiments Amazon S3 pour le compte.
- Ajouter une politique pour permettre à l'utilisateur d'assumer le rôle.
- Assumez le rôle et répertorier les compartiments S3 à l'aide d'informations d'identification temporaires, puis nettoyez les ressources.

SDK pour .NET

i Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

namespace IAMActions;

public class IAMWrapper
```

```

{
    private readonly IAMIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAMIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });
    }
}

```

```

        return response.AccessKey;
    }

    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

```

```

    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }

    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }

    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>

```

```

    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>

```

```

    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

```

    }

    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }

    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

```



```

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }

    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }

    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }

    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
    {

```

```
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }

    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }

    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
```

```
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }

    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
        ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }

    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
        ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }
}
```

```

    }

    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }

    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }

    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)

```

```

{
    var request = new PutRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutRolePolicyAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Add or update an inline policy document that is embedded in an IAM user.
/// </summary>
/// <param name="userName">The name of the IAM user.</param>
/// <param name="policyName">The name of the IAM policy.</param>
/// <param name="policyDocument">The policy document defining the IAM policy.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
string policyDocument)
{
    var request = new PutUserPolicyRequest
    {
        UserName = userName,
        PolicyName = policyName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutUserPolicyAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Wait for a new access key to be ready to use.
/// </summary>
/// <param name="accessKeyId">The Id of the access key.</param>
/// <returns>A boolean value indicating the success of the action.</returns>
public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
{
    var keyReady = false;

```

```

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}

```

```

using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<IAMWrapper>()
                    .AddTransient<UIWrapper>())

```

```

    )
    .Build();

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<IAMBasics>();

    IConfiguration configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load test settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

    // Values needed for user, role, and policies.
    string userName = configuration["UserName"]!;
    string s3PolicyName = configuration["S3PolicyName"]!;
    string roleName = configuration["RoleName"]!;

    var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
    var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

    uiWrapper.DisplayBasicsOverview();
    uiWrapper.PressEnter();

    // First create a user. By default, the new user has
    // no permissions.
    uiWrapper.DisplayTitle("Create User");
    Console.WriteLine($"Creating a new user with user name: {userName}.");
    var user = await iamWrapper.CreateUserAsync(userName);
    var userArn = user.Arn;

    Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
    uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

    // Define a role policy document that allows the new user
    // to assume the role.
    string assumeRolePolicyDocument = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": {" +

```

```

        $" \"AWS\": \"{userArn}\"" +
        "}," +
        "\"Action\": \"sts:AssumeRole\"" +
        "}]\" +
        "\"";

// Permissions to list all buckets.
string policyDocument = "{" +
    "\"Version\": \"2012-10-17\"," +
    "\"Statement\" : [{" +
        "\"Action\" : [\"s3:ListAllMyBuckets\"]," +
        "\"Effect\" : \"Allow\"," +
        "\"Resource\" : \"*\"," +
    "}]\" +
    "\"";

// Create an AccessKey for the user.
uiWrapper.DisplayTitle("Create access key");
Console.WriteLine("Now let's create an access key for the new user.");
var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

var accessKeyId = accessKey.AccessKeyId;
var secretAccessKey = accessKey.SecretAccessKey;

Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

// Now try listing the Amazon Simple Storage Service (Amazon S3)
// buckets. This should fail at this point because the user doesn't
// have permissions to perform this task.
uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
Console.WriteLine("Now let's try to display a list of the user's Amazon S3
buckets.");
var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
var buckets = await s3Wrapper.ListMyBucketsAsync();

```



```
Console.WriteLine(buckets is null
    ? "As expected, the call to list the buckets has returned a null list."
    : "Something went wrong. This shouldn't have worked.");

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Create IAM role");
Console.WriteLine($"Creating the role: {roleName}");

// Creating an IAM role to allow listing the S3 buckets. A role name
// is not case sensitive and must be unique to the account for which it
// is created.
var roleArn = await iamWrapper.CreateRoleAsync(roleName,
assumeRolePolicyDocument);

uiWrapper.PressEnter();

// Create a policy with permissions to list S3 buckets.
uiWrapper.DisplayTitle("Create IAM policy");
Console.WriteLine($"Creating the policy: {s3PolicyName}");
Console.WriteLine("with permissions to list the Amazon S3 buckets for the
account.");
var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
policyDocument);

// Wait 15 seconds for the IAM policy to be available.
uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

// Attach the policy to the role you created earlier.
uiWrapper.DisplayTitle("Attach new IAM policy");
Console.WriteLine("Now let's attach the policy to the role.");
await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

// Wait 15 seconds for the role to be updated.
Console.WriteLine();
uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

// Use the AWS Security Token Service (AWS STS) to have the user
// assume the role we created.
var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

// Wait for the new credentials to become valid.
uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");
```

```
var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

// Try again to list the buckets using the client created with
// the new user's credentials. This time, it should work.
var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

s3Wrapper.UpdateClients(s3Client2, stsClient2);

buckets = await s3Wrapper.ListMyBucketsAsync();

uiWrapper.DisplayTitle("List Amazon S3 buckets");
Console.WriteLine("This time we should have buckets to list.");
if (buckets is not null)
{
    buckets.ForEach(bucket =>
    {
        Console.WriteLine($"{bucket.BucketName} created:
{bucket.CreationDate}");
    });
}

uiWrapper.PressEnter();

// Now clean up all the resources used in the example.
uiWrapper.DisplayTitle("Clean up resources");
Console.WriteLine("Thank you for watching. The IAM Basics demo is
complete.");
Console.WriteLine("Please wait while we clean up the resources we
created.");

await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

await iamWrapper.DeletePolicyAsync(policy.Arn);

await iamWrapper.DeleteRoleAsync(roleName);

await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

await iamWrapper.DeleteUserAsync(userName);

uiWrapper.PressEnter();
```

```

        Console.WriteLine("All done cleaning up our resources. Thank you for your
patience.");
    }
}

namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {

```

```

        RoleSessionName = roleSession,
        RoleArn = roleToAssume,
    };

    var response = await _stsService.AssumeRoleAsync(request);

    return response.Credentials;
}

/// <summary>
/// Delete an S3 bucket.
/// </summary>
/// <param name="bucketName">Name of the S3 bucket to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
    { BucketName = bucketName });
    return result.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// List the buckets that are owned by the user's account.
/// </summary>
/// <returns>Async Task.</returns>
public async Task<List<S3Bucket>?> ListMyBucketsAsync()
{
    try
    {
        // Get the list of buckets accessible by the new user.
        var response = await _s3Service.ListBucketsAsync();

        return response.Buckets;
    }
    catch (AmazonS3Exception ex)
    {
        // Something else went wrong. Display the error message.
        Console.WriteLine($"Error: {ex.Message}");
        return null;
    }
}

/// <summary>

```

```

    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
    { BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}

namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
    }
}

```

```

        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
(IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.WriteLine("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }
}

```

```
/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter">The string to be centered.</param>
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title, and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);

    // Wait for the requested number of seconds.
    for (int i = numSeconds; i > 0; i--)
    {
        System.Threading.Thread.Sleep(1000);
        Console.Write($"{i}...");
    }

    PressEnter();
}
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Actions

AttachRolePolicy

L'exemple de code suivant montre comment utiliser `AttachRolePolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Attach an IAM policy to a role.
/// </summary>
/// <param name="policyArn">The policy to attach.</param>
/// <param name="roleName">The role that the policy will be attached to.</param>
```



```
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [AttachRolePolicy](#) à la section Référence des AWS SDK pour .NET API.

CreateAccessKey

L'exemple de code suivant montre comment utiliser `CreateAccessKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an IAM access key for a user.
/// </summary>
/// <param name="userName">The username for which to create the IAM access
/// key.</param>
/// <returns>The AccessKey.</returns>
public async Task<AccessKey> CreateAccessKeyAsync(string userName)
{
    var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
    {
```

```

        UserName = userName,
    });

    return response.AccessKey;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateAccessKey](#) à la section Référence des AWS SDK pour .NET API.

CreateInstanceProfile

L'exemple de code suivant montre comment utiliser `CreateInstanceProfile`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
    /// specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance. The role has attached policies that specify the AWS permissions
    /// granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
    /// role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,

```

```

    string roleName,
    string profileName,
    string ssmOnlyPolicyFile,
    List<string>? awsManagedPolicies = null)
{

    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", \" +
        "\"Statement\": [{\" +
            "\"Effect\": \"Allow\", \" +
            "\"Principal\": {\" +
            "\"Service\": [\" +
                \"ec2.amazonaws.com\"\" +
            "]" +
            "}, \" +
            "\"Action\": \"sts:AssumeRole\"\" +
            "]" +
        "}";

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {
        var createPolicyResult = await _amazonIam.CreatePolicyAsync(
            new CreatePolicyRequest
            {
                PolicyName = policyName,
                PolicyDocument = policyDocument
            });
        policyArn = createPolicyResult.Policy.Arn;
    }
    catch (EntityAlreadyExistsException)
    {
        // The policy already exists, so we look it up to get the Arn.
        var policiesPaginator = _amazonIam.Paginators.ListPolicies(
            new ListPoliciesRequest()
            {
                Scope = PolicyScopeType.Local
            });
        // Get the entire list using the paginator.
        await foreach (var policy in policiesPaginator.Policies)
        {

```

```

        if (policy.PolicyName.Equals(policyName))
        {
            policyArn = policy.Arn;
        }
    }

    if (policyArn == null)
    {
        throw new InvalidOperationException("Policy not found");
    }
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try

```

```

    {
        var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
            new CreateInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        // Allow time for the profile to be ready.
        profileArn = profileCreateResponse.InstanceProfile.Arn;
        Thread.Sleep(10000);
        await _amazonIam.AddRoleToInstanceProfileAsync(
            new AddRoleToInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy already exists.");
        var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
            new GetInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateInstanceProfile](#) à la section Référence des AWS SDK pour .NET API.

CreatePolicy

L'exemple de code suivant montre comment utiliser `CreatePolicy`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an IAM policy.
/// </summary>
/// <param name="policyName">The name to give the new IAM policy.</param>
/// <param name="policyDocument">The policy document for the new policy.</param>
/// <returns>The new IAM policy object.</returns>
public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
{
    var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
    {
        PolicyDocument = policyDocument,
        PolicyName = policyName,
    });

    return response.Policy;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreatePolicy](#) à la section Référence des AWS SDK pour .NET API.

CreateRole

L'exemple de code suivant montre comment utiliser `CreateRole`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role.</param>
/// <param name="rolePolicyDocument">The name of the IAM policy document
/// for the new role.</param>
/// <returns>The Amazon Resource Name (ARN) of the role.</returns>
public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
{
    var request = new CreateRoleRequest
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = rolePolicyDocument,
    };

    var response = await _IAMService.CreateRoleAsync(request);
    return response.Role.Arn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateRole](#) à la section Référence des AWS SDK pour .NET API.

CreateServiceLinkedRole

L'exemple de code suivant montre comment utiliser `CreateServiceLinkedRole`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an IAM service-linked role.
/// </summary>
/// <param name="serviceName">The name of the AWS Service.</param>
/// <param name="description">A description of the IAM service-linked role.</
param>
/// <returns>The IAM role that was created.</returns>
public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
description)
{
    var request = new CreateServiceLinkedRoleRequest
    {
        AWSServiceName = serviceName,
        Description = description
    };

    var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
    return response.Role;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateServiceLinkedRole](#) à la section Référence des AWS SDK pour .NET API.

CreateUser

L'exemple de code suivant montre comment utiliser CreateUser.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an IAM user.
/// </summary>
/// <param name="userName">The username for the new IAM user.</param>
/// <returns>The IAM user that was created.</returns>
public async Task<User> CreateUserAsync(string userName)
{
    var response = await _IAMService.CreateUserAsync(new CreateUserRequest
    { Username = userName });
    return response.User;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateUser](#) à la section Référence des AWS SDK pour .NET API.

DeleteAccessKey

L'exemple de code suivant montre comment utiliser `DeleteAccessKey`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an IAM user's access key.
/// </summary>
```

```

    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
    userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
    DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

- Pour plus de détails sur l'API, reportez-vous [DeleteAccessKey](#) à la section Référence des AWS SDK pour .NET API.

DeleteInstanceProfile

L'exemple de code suivant montre comment utiliser `DeleteInstanceProfile`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>

```

```
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                });
            // Delete the custom policies only.
            if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
            {
                await _amazonIam.DeletePolicyAsync(
                    new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                    {
                        PolicyArn = policy.PolicyArn
                    });
            }
        }

        await _amazonIam.DeleteRoleAsync(
            new DeleteRoleRequest() { RoleName = roleName });
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine($"Instance profile {profileName} does not exist.");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteInstanceProfile](#) à la section Référence des AWS SDK pour .NET API.

DeletePolicy

L'exemple de code suivant montre comment utiliser `DeletePolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an IAM policy.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
/// delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeletePolicyAsync(string policyArn)
{
    var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
    { PolicyArn = policyArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeletePolicy](#) à la section Référence des AWS SDK pour .NET API.

DeleteRole

L'exemple de code suivant montre comment utiliser `DeleteRole`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteRoleAsync(string roleName)
{
    var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
    { RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteRole](#) à la section Référence des AWS SDK pour .NET API.

DeleteRolePolicy

L'exemple de code suivant montre comment utiliser DeleteRolePolicy.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an IAM role policy.
```

```

    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

- Pour plus de détails sur l'API, reportez-vous [DeleteRolePolicy](#) à la section Référence des AWS SDK pour .NET API.

DeleteUser

L'exemple de code suivant montre comment utiliser `DeleteUser`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {

```

```
var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteUser](#) à la section Référence des AWS SDK pour .NET API.

DeleteUserPolicy

L'exemple de code suivant montre comment utiliser `DeleteUserPolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an IAM user policy.
/// </summary>
/// <param name="policyName">The name of the IAM policy to delete.</param>
/// <param name="userName">The username of the IAM user.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
{
    var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteUserPolicy](#) à la section Référence des AWS SDK pour .NET API.

DetachRolePolicy

L'exemple de code suivant montre comment utiliser `DetachRolePolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Detach an IAM policy from an IAM role.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
/// <param name="roleName">The name of the IAM role.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
{
    var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
    {
        PolicyArn = policyArn,
        RoleName = roleName,
    });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DetachRolePolicy](#) à la section Référence des AWS SDK pour .NET API.

GetAccountPasswordPolicy

L'exemple de code suivant montre comment utiliser `GetAccountPasswordPolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Gets the IAM password policy for an AWS account.
/// </summary>
/// <returns>The PasswordPolicy for the AWS account.</returns>
public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
{
    var response = await _IAMService.GetAccountPasswordPolicyAsync(new
    GetAccountPasswordPolicyRequest());
    return response.PasswordPolicy;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetAccountPasswordPolicy](#) à la section Référence des AWS SDK pour .NET API.

GetPolicy

L'exemple de code suivant montre comment utiliser `GetPolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get information about an IAM policy.
/// </summary>
/// <param name="policyArn">The IAM policy to retrieve information for.</param>
/// <returns>The IAM policy.</returns>
public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
{
    var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
    { PolicyArn = policyArn });
    return response.Policy;
}

```

- Pour plus de détails sur l'API, reportez-vous [GetPolicy](#) à la section Référence des AWS SDK pour .NET API.

GetRole

L'exemple de code suivant montre comment utiliser `GetRole`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,

```

```
});  
  
    return response.Role;  
}
```

- Pour plus de détails sur l'API, reportez-vous [GetRole](#) à la section Référence des AWS SDK pour .NET API.

GetUser

L'exemple de code suivant montre comment utiliser `GetUser`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Get information about an IAM user.  
/// </summary>  
/// <param name="userName">The username of the user.</param>  
/// <returns>An IAM user object.</returns>  
public async Task<User> GetUserAsync(string userName)  
{  
    var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName  
= userName });  
    return response.User;  
}
```

- Pour plus de détails sur l'API, reportez-vous [GetUser](#) à la section Référence des AWS SDK pour .NET API.

ListAttachedRolePolicies

L'exemple de code suivant montre comment utiliser `ListAttachedRolePolicies`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List the IAM role policies that are attached to an IAM role.
/// </summary>
/// <param name="roleName">The IAM role to list IAM policies for.</param>
/// <returns>A list of the IAM policies attached to the IAM role.</returns>
public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
{
    var attachedPolicies = new List<AttachedPolicyType>();
    var attachedRolePoliciesPaginator =
        _IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
        { RoleName = roleName });

    await foreach (var response in attachedRolePoliciesPaginator.Responses)
    {
        attachedPolicies.AddRange(response.AttachedPolicies);
    }

    return attachedPolicies;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListAttachedRolePolicies](#) à la section Référence des AWS SDK pour .NET API.

ListGroups

L'exemple de code suivant montre comment utiliser `ListGroups`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List IAM groups.
/// </summary>
/// <returns>A list of IAM groups.</returns>
public async Task<List<Group>> ListGroupsAsync()
{
    var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
    var groups = new List<Group>();

    await foreach (var response in groupsPaginator.Responses)
    {
        groups.AddRange(response.Groups);
    }

    return groups;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListGroups](#) à la section Référence des AWS SDK pour .NET API.

ListPolicies

L'exemple de code suivant montre comment utiliser `ListPolicies`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List IAM policies.
/// </summary>
/// <returns>A list of the IAM policies.</returns>
public async Task<List<ManagedPolicy>> ListPoliciesAsync()
{
    var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
    var policies = new List<ManagedPolicy>();

    await foreach (var response in listPoliciesPaginator.Responses)
    {
        policies.AddRange(response.Policies);
    }

    return policies;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListPolicies](#) à la section Référence des AWS SDK pour .NET API.

ListRolePolicies

L'exemple de code suivant montre comment utiliser `ListRolePolicies`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List IAM role policies.
/// </summary>
/// <param name="roleName">The IAM role for which to list IAM policies.</param>
/// <returns>A list of IAM policy names.</returns>
public async Task<List<string>> ListRolePoliciesAsync(string roleName)
{
    var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
    var policyNames = new List<string>();

    await foreach (var response in listRolePoliciesPaginator.Responses)
    {
        policyNames.AddRange(response.PolicyNames);
    }

    return policyNames;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListRolePolicies](#) à la section Référence des AWS SDK pour .NET API.

ListRoles

L'exemple de code suivant montre comment utiliser `ListRoles`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List IAM roles.
/// </summary>
/// <returns>A list of IAM roles.</returns>
public async Task<List<Role>> ListRolesAsync()
{
    var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
    var roles = new List<Role>();

    await foreach (var response in listRolesPaginator.Responses)
    {
        roles.AddRange(response.Roles);
    }

    return roles;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListRoles](#) à la section Référence des AWS SDK pour .NET API.

ListSAMLProviders

L'exemple de code suivant montre comment utiliser `ListSAMLProviders`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List SAML authentication providers.
/// </summary>
/// <returns>A list of SAML providers.</returns>
public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
{
    var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
    return response.SAMLProviderList;
}
```

- Pour plus de détails sur l'API, consultez [la section Liste SAMLProviders](#) dans la référence des AWS SDK pour .NET API.

ListUsers

L'exemple de code suivant montre comment utiliser `ListUsers`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List IAM users.
/// </summary>
/// <returns>A list of IAM users.</returns>
```

```

public async Task<List<User>> ListUsersAsync()
{
    var listUsersPaginator = _IAMService.Paginators.ListUsers(new
ListUsersRequest());
    var users = new List<User>();

    await foreach (var response in listUsersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}

```

- Pour plus de détails sur l'API, reportez-vous [ListUsers](#) à la section Référence des AWS SDK pour .NET API.

PutRolePolicy

L'exemple de code suivant montre comment utiliser PutRolePolicy.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Update the inline policy document embedded in a role.
/// </summary>
/// <param name="policyName">The name of the policy to embed.</param>
/// <param name="roleName">The name of the role to update.</param>
/// <param name="policyDocument">The policy document that defines the role.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
string policyDocument)

```

```
{
    var request = new PutRolePolicyRequest
    {
        PolicyName = policyName,
        RoleName = roleName,
        PolicyDocument = policyDocument
    };

    var response = await _IAMService.PutRolePolicyAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [PutRolePolicy](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer et gérer un service résilient

L'exemple de code suivant montre comment créer un service Web à charge équilibrée qui renvoie des recommandations de livres, de films et de chansons. L'exemple montre comment le service répond aux défaillances et comment le restructurer pour accroître la résilience en cas de défaillance.

- Utilisez un groupe Amazon EC2 Auto Scaling pour créer des instances Amazon Elastic Compute Cloud (Amazon EC2) sur la base d'un modèle de lancement et pour maintenir le nombre d'instances dans une plage spécifiée.
- Gérez et distribuez les requêtes HTTP avec Elastic Load Balancing.
- Surveillez l'état des instances d'un groupe Auto Scaling et transférez les demandes uniquement aux instances saines.
- Exécutez un serveur Web Python sur chaque EC2 instance pour gérer les requêtes HTTP. Le serveur Web répond par des recommandations et des surveillances de l'état.
- Simulez un service de recommandation avec une table Amazon DynamoDB.
- Contrôlez la réponse du serveur Web aux demandes et aux contrôles de santé en mettant à jour AWS Systems Manager les paramètres.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    // Set up dependency injection for the AWS services.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
                .AddAWSService<IAmazonDynamoDB>()
                .AddAWSService<IAmazonElasticLoadBalancingV2>()
                .AddAWSService<IAmazonSimpleSystemsManagement>()
                .AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalerWrapper>()
                .AddTransient<ElasticLoadBalancerWrapper>()
                .AddTransient<SmParameterWrapper>()
                .AddTransient<Recommendations>()
                .AddSingleton<IConfiguration>(_configuration)
            )
        .Build();

    ServicesSetup(host);
}
```

```

ResourcesSetup();

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
    Console.WriteLine(new string('-', 80));
    await Deploy(true);

    Console.WriteLine("Now let's begin the scenario.");
    Console.WriteLine(new string('-', 80));
    await Demo(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Finally, let's clean up our resources.");
    Console.WriteLine(new string('-', 80));

    await DestroyResources(true);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
    Console.WriteLine(new string('-', 80));
}
catch (Exception ex)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
    await DestroyResources(true);
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>
/// Setup any common resources, also used for integration testing.
/// </summary>
public static void ResourcesSetup()
{
    _httpClient = new HttpClient();
}

/// <summary>

```

```

    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(

```

```

        "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");

```

```

        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

```



```

        await
        _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadBalancerName,
        subnetIds, targetGroup);
        await
        _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
        targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
        _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        var loadBalancerAccess = await
        _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
            that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
            checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
            _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
            _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
            ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
                    default VPC must\n"
                    + "allows access from this computer. You can either add it
                    automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
                    \n");

                if (!interactive || GetYesNoResponse(
                    "Do you want to add a rule to the security group to allow
                    inbound traffic from your computer's IP address?"))
                {

```

```

        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
    }
}

if (!sshPortIsOpen)
{
    if (!interactive || GetYesNoResponse(
        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
    {
        await
        _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
    }
}

loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

if (loadBalancerAccess)
{
    Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
    Console.WriteLine($"\\thttp://{endPoint}\\n");
}
else
{
    Console.WriteLine(
        "\\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\\n"
        + "manually verifying that your VPC and security group are
configured correctly and that\\n"
        + "you can successfully make a GET request to the load balancer
endpoint:\\n");
    Console.WriteLine($"\\thttp://{endPoint}\\n");
}
Console.WriteLine(new string('-', 80));
Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
if (interactive)
    Console.ReadLine();
return true;
}

```

```

/// <summary>
/// Demonstrate the steps of the scenario.
/// </summary>
/// <param name="interactive">True to run as an interactive scenario.</param>
/// <returns>Async task.</returns>
public static async Task<bool> Demo(bool interactive)
{
    var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
        "ssm_only_policy.json");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Resetting parameters to starting values for demo.");
    await _smParameterWrapper.Reset();

    Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
        "to create situations where the web service fails, and
shows how using a resilient\n" +
        "architecture can keep the web service running in spite of
these failures.");
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
        $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
        $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
    Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a failure code. But, the service reports as\n" +
        "healthy to the load balancer because shallow health
checks don't check for failure of the recommendation service.");
    if (interactive)
        await DemoActionChoices();
}

```

```

        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,

```

```

        _autoScalerWrapper.BadCredsProfileName,
        instanceProfile.AssociationId
    );
    Console.WriteLine(
        "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
        "depending on which instance is selected by the load balancer.\n"
    );
    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
    Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
    Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
    Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
    Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

    Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
    Console.WriteLine("and take that instance out of rotation.");

    await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

    Console.WriteLine($"Now, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
    Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
    Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
    Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

    if (interactive)
        await DemoActionChoices();

    Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");

```

```

        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\\nWhen all instances are unhealthy, the load balancer
continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>

```

```

/// <returns>Async task.</returns>
public static async Task<bool> DestroyResources(bool interactive)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
        "that were created for this demo."
    );

    if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
    {
        await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
        await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
        await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
        await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
        await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
        await _autoScalerWrapper.DeleteInstanceProfile(
            _autoScalerWrapper.BadCredsProfileName,
            _autoScalerWrapper.BadCredsRoleName
        );
        await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
    }
    else
    {
        Console.WriteLine(
            "Ok, we'll leave the resources intact.\n" +
            "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
        );
    }

    Console.WriteLine(new string('-', 80));
    return true;
}

```

Créez une classe qui englobe les EC2 actions Auto Scaling et Amazon.

```

/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;
    private readonly ILogger<AutoScalerWrapper> _logger;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,

```



```

    IAmazonIdentityManagementService amazonIam,
    IConfiguration configuration,
    ILogger<AutoScalerWrapper> logger)
{
    _amazonAutoScaling = amazonAutoScaling;
    _amazonEc2 = amazonEc2;
    _amazonSsm = amazonSsm;
    _amazonIam = amazonIam;
    _logger = logger;

    var prefix = configuration["resourcePrefix"];
    _instanceType = configuration["instanceType"];
    _amiParam = configuration["amiParam"];

    _launchTemplateName = prefix + "-template";
    _groupName = prefix + "-group";
    _instancePolicyName = prefix + "-pol";
    _instanceRoleName = prefix + "-role";
    _instanceProfileName = prefix + "-prof";
    _badCredsPolicyName = prefix + "-bc-pol";
    _badCredsRoleName = prefix + "-bc-role";
    _badCredsProfileName = prefix + "-bc-prof";
    _keyPairName = prefix + "-key-pair";
}

/// <summary>
/// Create a policy, role, and profile that is associated with instances with a
specified name.
/// An instance's associated profile defines a role that is assumed by the
/// instance. The role has attached policies that specify the AWS permissions
granted to
/// clients that run on the instance.
/// </summary>
/// <param name="policyName">Name to use for the policy.</param>
/// <param name="roleName">Name to use for the role.</param>
/// <param name="profileName">Name to use for the profile.</param>
/// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
/// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
/// <returns>The Arn of the profile.</returns>
public async Task<string> CreateInstanceProfileWithName(
    string policyName,
    string roleName,
    string profileName,

```

```

string ssmOnlyPolicyFile,
List<string>? awsManagedPolicies = null)
{

    var assumeRoleDoc = "{" +
        "\"Version\": \"2012-10-17\", \" +
        "\"Statement\": [{\" +
            "\"Effect\": \"Allow\", \" +
            "\"Principal\": {\" +
            "\"Service\": [\" +
                \"ec2.amazonaws.com\"\" +
            "]" +
            \",\" +
            "\"Action\": \"sts:AssumeRole\"\" +
            \"]}\" +
        "};

    var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

    var policyArn = "";

    try
    {
        var createPolicyResult = await _amazonIam.CreatePolicyAsync(
            new CreatePolicyRequest
            {
                PolicyName = policyName,
                PolicyDocument = policyDocument
            });
        policyArn = createPolicyResult.Policy.Arn;
    }
    catch (EntityAlreadyExistsException)
    {
        // The policy already exists, so we look it up to get the Arn.
        var policiesPaginator = _amazonIam.Paginators.ListPolicies(
            new ListPoliciesRequest()
            {
                Scope = PolicyScopeType.Local
            });
        // Get the entire list using the paginator.
        await foreach (var policy in policiesPaginator.Policies)
        {
            if (policy.PolicyName.Equals(policyName))
            {

```

```

        policyArn = policy.Arn;
    }
}

if (policyArn == null)
{
    throw new InvalidOperationException("Policy not found");
}

try
{
    await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
    {
        RoleName = roleName,
        AssumeRolePolicyDocument = assumeRoleDoc,
    });
    await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
    {
        RoleName = roleName,
        PolicyArn = policyArn
    });
    if (awsManagedPolicies != null)
    {
        foreach (var awsPolicy in awsManagedPolicies)
        {
            await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
            {
                PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                RoleName = roleName
            });
        }
    }
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("Role already exists.");
}

string profileArn = "";
try
{
    var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(

```

```

        new CreateInstanceProfileRequest()
        {
            InstanceProfileName = profileName
        });
    // Allow time for the profile to be ready.
    profileArn = profileCreateResponse.InstanceProfile.Arn;
    Thread.Sleep(10000);
    await _amazonIam.AddRoleToInstanceProfileAsync(
        new AddRoleToInstanceProfileRequest()
        {
            InstanceProfileName = profileName,
            RoleName = roleName
        });

    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy already exists.");
        var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
            new GetInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}

/// <summary>
/// Create a new key pair and save the file.
/// </summary>
/// <param name="newKeyPairName">The name of the new key pair.</param>
/// <returns>Async task.</returns>
public async Task CreateKeyPair(string newKeyPairName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyPairName });
        await File.WriteAllTextAsync($"{newKeyPairName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyPairName}.");
    }
    catch (AlreadyExistsException)

```

```

        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
    after
    /// the instance is started. This script installs the Python packages and starts
    a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
    run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
    and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
    startupScriptPath, string instancePolicyPath)
    {
        try
        {
            await CreateKeyPair(_keyPairName);

```

```

        await CreateInstanceProfileWithName(_instancePolicyName,
        _instanceRoleName,
            _instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes =
            System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
            "InvalidLaunchTemplateName.AlreadyExistsException")
        {
            _logger.LogError($"Could not create the template, the name
{_launchTemplateName} already exists. " +
                $"Please try again with a unique name.");
        }

        throw;
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        try
        {
            var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
                new DescribeAvailabilityZonesRequest());
            return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            _logger.LogError($"An Amazon EC2 error occurred while listing
availability zones.: {ec2Exception.Message}");
            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while listing availability zones.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)

```

```

    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        try
        {
            var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
                new DescribeVpcsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("is-default", new List<string>() { "true" })
                    }
                });
            return vpcResponse.Vpcs[0];
        }
    }

```



```

    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "UnauthorizedOperation")
        {
            _logger.LogError(ec2Exception, $"You do not have the necessary
permissions to describe VPCs.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the vpcs.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Get all the subnets for a Vpc in a set of availability zones.
/// </summary>
/// <param name="vpcId">The Id of the Vpc.</param>
/// <param name="availabilityZones">The list of availability zones.</param>
/// <returns>The collection of subnet objects.</returns>
public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
{
    try
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new("vpc-id", new List<string>() { vpcId }),
                    new("availability-zone", availabilityZones),
                    new("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)

```

```

        {
            subnets.Add(subnet);
        }

        return subnets;
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidVpcID.NotFound")
        {
            _logger.LogError(ec2Exception, $"The specified VPC ID {vpcId} does
not exist.");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while describing the subnets.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode ==
"InvalidLaunchTemplateName.NotFoundException")
        {

```

```

        _logger.LogError(
            $"Could not delete the template, the name {_launchTemplateName}
was not found.");
    }

    throw;
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while deleting the template.:
{ex.Message}");
    throw;
}
}

/// <summary>
/// Detaches a role from an instance profile, detaches policies from the role,
/// and deletes all the resources.
/// </summary>
/// <param name="profileName">The name of the profile to delete.</param>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteInstanceProfile(string profileName, string roleName)
{
    try
    {
        await _amazonIam.RemoveRoleFromInstanceProfileAsync(
            new RemoveRoleFromInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });
        await _amazonIam.DeleteInstanceProfileAsync(
            new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
        var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
            new ListAttachedRolePoliciesRequest() { RoleName = roleName });
        foreach (var policy in attachedPolicies.AttachedPolicies)
        {
            await _amazonIam.DetachRolePolicyAsync(
                new DetachRolePolicyRequest()
                {
                    RoleName = roleName,
                    PolicyArn = policy.PolicyArn
                }
            );
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the instance profile.:
{ex.Message}");
        throw;
    }
}

```

```

        });
        // Delete the custom policies only.
        if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
        {
            await _amazonIam.DeletePolicyAsync(
                new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                {
                    PolicyArn = policy.PolicyArn
                });
        }
    }

    await _amazonIam.DeleteRoleAsync(
        new DeleteRoleRequest() { RoleName = roleName });
}
catch (NoSuchEntityException)
{
    Console.WriteLine($"Instance profile {profileName} does not exist.");
}
}

/// <summary>
/// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
/// </summary>
/// <param name="group">The name of the auto scaling group.</param>
/// <returns>A collection of instance Ids.</returns>
public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
{
    var instanceResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
    var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
        g => g.Instances.Select(i => i.InstanceId));
    return instanceIds;
}

/// <summary>
/// Get the instance profile association data for an instance.
/// </summary>
/// <param name="instanceId">The Id of the instance.</param>

```

```

    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        try
        {
            var response = await
_amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
                new DescribeIamInstanceProfileAssociationsRequest()
                {
                    Filters = new List<Amazon.EC2.Model.Filter>()
                    {
                        new("instance-id", new List<string>() { instanceId })
                    },
                });
            return response.IamInstanceProfileAssociations[0];
        }
        catch (AmazonEC2Exception ec2Exception)
        {
            if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
            {
                _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
            }

            throw;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while creating the template.:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>

```

```

    /// <param name="associationId">The Id of the existing profile association for
    the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
    credsProfileName, string associationId)
    {
        try
        {
            await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
            // Allow time before resetting.
            Thread.Sleep(25000);

            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(25000);
            var instanceReady = false;
            var retries = 5;
            while (retries-- > 0 && !instanceReady)
            {
                var instancesPaginator =
                    _amazonSsm.Paginators.DescribeInstanceInformation(
                        new DescribeInstanceInformationRequest());
                // Get the entire list using the paginator.
                await foreach (var instance in
instancesPaginator.InstanceInformationList)
                {
                    instanceReady = instance.InstanceId == instanceId;
                    if (instanceReady)
                    {
                        break;
                    }
                }
            }
            Console.WriteLine("Waiting for instance to be running.");
            await WaitForInstanceState(instanceId, InstanceStateName.Running);
            Console.WriteLine("Instance ready.");

```

```

        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                    {
                        "commands",
                        new List<string>() { "cd / && sudo python3 server.py"
80" }
                    }
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
    catch (AmazonEC2Exception ec2Exception)
    {
        if (ec2Exception.ErrorCode == "InvalidInstanceID.NotFound")
        {
            _logger.LogError(ec2Exception, $"Instance {instanceId} not found");
        }

        throw;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"An error occurred while replacing the template.:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Try to terminate an instance by its Id.
/// </summary>
/// <param name="instanceId">The Id of the instance to terminate.</param>
/// <returns>Async task.</returns>
public async Task TryTerminateInstanceById(string instanceId)
{
    var stopping = false;
    Console.WriteLine($"Stopping {instanceId}...");
    while (!stopping)

```

```

        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
    progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                    || (e is Amazon.AutoScaling.Model.ResourceInUseException))
        }
    }

```



```

        {
            Console.WriteLine($"Some instances are still running. Waiting...");
            Thread.Sleep(10000);
        }
    }
}

/// <summary>
/// Terminate instances and delete the Auto Scaling group by name.
/// </summary>
/// <param name="groupName">The name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
{
    var describeGroupsResponse = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
    if (describeGroupsResponse.AutoScalingGroups.Any())
    {
        // Update the size to 0.
        await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
            new UpdateAutoScalingGroupRequest()
            {
                AutoScalingGroupName = groupName,
                MinSize = 0
            });
        var group = describeGroupsResponse.AutoScalingGroups[0];
        foreach (var instance in group.Instances)
        {
            await TryTerminateInstanceById(instance.InstanceId);
        }

        await TryDeleteGroupByName(groupName);
    }
    else
    {
        Console.WriteLine($"No groups found with name {groupName}.");
    }
}

```

```

    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
    computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
    instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
    while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;

```

```

        if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
        {
            portIsOpen = true;
        }
    }

    if (ipPermission.PrefixListIds.Any())
    {
        portIsOpen = true;
    }

    if (!portIsOpen)
    {
        Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
    }
    else
    {
        break;
    }
}

return portIsOpen;
}

/// <summary>
/// Add an ingress rule to the specified security group that allows access on
the
/// specified port from the specified IP address.
/// </summary>
/// <param name="groupId">The Id of the security group to modify.</param>
/// <param name="port">The port to open.</param>
/// <param name="ipAddress">The IP address to allow access.</param>
/// <returns>Async task.</returns>
public async Task OpenInboundPort(string groupId, int port, string ipAddress)
{
    await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
        new AuthorizeSecurityGroupIngressRequest()
        {
            GroupId = groupId,
            IpPermissions = new List<IpPermission>()

```

```

        {
            new IpPermission()
            {
                FromPort = port,
                ToPort = port,
                IpProtocol = "tcp",
                Ipv4Ranges = new List<IpRange>()
                {
                    new IpRange() { CidrIp = $"{ipAddress}/32" }
                }
            }
        }
    });
}

/// <summary>
/// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
/// The
/// </summary>
/// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
/// <param name="targetGroupArn">The Arn for the target group.</param>
/// <returns>Async task.</returns>
public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
{
    await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
        new AttachLoadBalancerTargetGroupsRequest()
        {
            AutoScalingGroupName = autoScalingGroupName,
            TargetGroupARNs = new List<string>() { targetGroupArn }
        });
}

/// <summary>
/// Wait until an EC2 instance is in a specified state.
/// </summary>
/// <param name="instanceId">The instance Id.</param>
/// <param name="stateName">The state to wait for.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
{

```

```

    var request = new DescribeInstancesRequest
    {
        InstanceIds = new List<string> { instanceId }
    };

    // Wait until the instance is in the specified state.
    var hasState = false;
    do
    {
        // Wait 5 seconds.
        Thread.Sleep(5000);

        // Check for the desired state.
        var response = await _amazonEc2.DescribeInstancesAsync(request);
        var instance = response.Reservations[0].Instances[0];
        hasState = instance.State.Name == stateName;
        Console.Write(". ");
    } while (!hasState);

    return hasState;
}
}

```

Créez une classe qui englobe les actions Elastic Load Balancing.

```

/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.

```

```

    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
    }

```

```

        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///

```

```

    /// To speed up this demo, the health check is configured with shortened times
    and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
    unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
        ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
        _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
    subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
        List<string> subnetIds, TargetGroup targetGroup)
    {

```



```
var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
    new CreateLoadBalancerRequest()
    {
        Name = name,
        Subnets = subnetIds
    });
var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

// Wait for load balancer to be available.
var loadBalancerReady = false;
while (!loadBalancerReady)
{
    try
    {
        var describeResponse =
            await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                new DescribeLoadBalancersRequest()
                {
                    Names = new List<string>() { name }
                });

        var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

        loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
    }
    catch (LoadBalancerNotFoundException)
    {
        loadBalancerReady = false;
    }
    Thread.Sleep(10000);
}
// Create the listener.
await _amazonElasticLoadBalancingV2.CreateListenerAsync(
    new CreateListenerRequest()
    {
        LoadBalancerArn = loadBalancerArn,
        Protocol = targetGroup.Protocol,
        Port = targetGroup.Port,
        DefaultActions = new List<Action>()
        {
            new Action()
```

```

        {
            Type = ActionTypeEnum.Forward,
            TargetGroupArn = targetGroup.TargetGroupArn
        }
    }
});
return createLbResponse.LoadBalancers[0];
}

/// <summary>
/// Verify this computer can successfully send a GET request to the
/// load balancer endpoint.
/// </summary>
/// <param name="endpoint">The endpoint to check.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
{
    var success = false;
    var retries = 3;
    while (!success && retries > 0)
    {
        try
        {
            var endpointResponse = await _httpClient.GetAsync($"http://{
{endpoint}");
            Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

            if (endpointResponse.IsSuccessStatusCode)
            {
                success = true;
            }
            else
            {
                retries = 0;
            }
        }
        catch (HttpRequestException)
        {
            Console.WriteLine("Connection error, retrying...");
            retries--;
            Thread.Sleep(10000);
        }
    }
}

```

```

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
                describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try

```

```

        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });

            var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
            await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
            Console.WriteLine($"Deleted load balancing target group
{groupName}.");
            done = true;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine(
                $"Target group {groupName} not found, could not delete.");
            done = true;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine("Target group not yet released, waiting...");
            Thread.Sleep(10000);
        }
    }
}
}

```

Créez une classe qui utilise DynamoDB pour simuler un service de recommandation.

```

/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
/// and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;
}

```

```

public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()

```

```

        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("\nWaiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = tableName
};

TableStatus status;
do
{
    Thread.Sleep(2000);

    var describeTableResponse = await
        _amazonDynamoDb.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException)
{
    Console.WriteLine($"Table {tableName} already exists.");
}

```

```

        return false;
    }
}

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
    var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
    var records =
        JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
    var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

    foreach (var record in records!)
    {
        batchWrite.AddPutItem(record);
    }

    await batchWrite.ExecuteAsync();
}

/// <summary>
/// Delete the recommendation table by name.
/// </summary>
/// <param name="tableName">The name of the recommendation table.</param>
/// <returns>Async task.</returns>
public async Task DestroyDatabaseByName(string tableName)
{
    try
    {
        await _amazonDynamoDb.DeleteTableAsync(
            new DeleteTableRequest() { TableName = tableName });
        Console.WriteLine($"Table {tableName} was deleted.");
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"Table {tableName} not found");
    }
}

```

```

    }
}

```

Créez une classe qui englobe les actions de Systems Manager.

```

/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
/// parameters
/// to drive the demonstration of resilient architecture, such as failure of a
/// dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }
}

```



```

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
true });
    }
}

```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)

- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Exemples d'utilisation d'Amazon Keyspaces SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Keyspaces.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour Amazon Keyspaces

L'exemple de code suivant montre comment commencer à utiliser Amazon Keyspaces.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace KeyspacesActions;

public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache Cassandra).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient = host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
```

```
        await keyspacesWrapper.ListKeyspaces();  
    }  
}
```

- Pour plus de détails sur l'API, reportez-vous [ListKeyspaces](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer un espace de clés et une table. Le schéma de table contient les données vidéo et la point-in-time restauration est activée.
- se connecter à l'espace de clés à l'aide d'une connexion TLS sécurisée avec authentification SigV4 ;
- interroger la table. Ajoutez, extrayez et mettez à jour les données des films ;
- mettre à jour la table. Ajoutez une colonne pour suivre les films visionnés ;
- restaurer l'état précédent de la table et nettoyer les ressources.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
global using System.Security.Cryptography.X509Certificates;  
global using Amazon.Keyspaces;  
global using Amazon.Keyspaces.Model;  
global using KeyspacesActions;  
global using KeyspacesScenario;  
global using Microsoft.Extensions.Configuration;  
global using Microsoft.Extensions.DependencyInjection;
```

```
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using Newtonsoft.Json;

namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                    .AddTransient<KeyspacesWrapper>()
                    .AddTransient<CassandraWrapper>()
                )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var keyspacesWrapper = host.Services.GetRequiredService<KeyspacesWrapper>();
```

```
var uiMethods = new UiMethods();

var keyspaceName = configuration["KeyspaceName"];
var tableName = configuration["TableName"];

bool success; // Used to track the results of some operations.

uiMethods.DisplayOverview();
uiMethods.PressEnter();

// Create the keyspace.
var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);

// Wait for the keyspace to be available. GetKeyspace results in a
// resource not found error until it is ready for use.
try
{
    var getKeySpaceArn = "";
    Console.WriteLine($"Created {keyspaceName}. Waiting for it to become
available. ");
    do
    {
        getKeySpaceArn = await keyspacesWrapper.GetKeyspace(keyspaceName);
        Console.WriteLine(". ");
    } while (getKeySpaceArn != keyspaceArn);
}
catch (ResourceNotFoundException)
{
    Console.WriteLine("Waiting for keyspace to be created.");
}

Console.WriteLine($"\\nThe keyspace {keyspaceName} is ready for use.");

uiMethods.PressEnter();

// Create the table.
// First define the schema.
var allColumns = new List<ColumnDefinition>
{
    new ColumnDefinition { Name = "title", Type = "text" },
    new ColumnDefinition { Name = "year", Type = "int" },
    new ColumnDefinition { Name = "release_date", Type = "timestamp" },
    new ColumnDefinition { Name = "plot", Type = "text" },
};
```

```

var partitionKeys = new List<PartitionKey>
{
    new PartitionKey { Name = "year", },
    new PartitionKey { Name = "title" },
};

var tableSchema = new SchemaDefinition
{
    AllColumns = allColumns,
    PartitionKeys = partitionKeys,
};

var tableArn = await keyspacesWrapper.CreateTable(keyspaceName, tableSchema,
tableName);

// Wait for the table to be active.
try
{
    var resp = new GetTableResponse();
    Console.WriteLine("Waiting for the new table to be active. ");
    do
    {
        try
        {
            resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
            Console.WriteLine(".");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine(".");
        }
    } while (resp.Status != TableStatus.ACTIVE);

    // Display the table's schema.
    Console.WriteLine($"\\nTable {tableName} has been created in
{keyspaceName}");
    Console.WriteLine("Let's take a look at the schema.");
    uiMethods.DisplayTitle("All columns");
    resp.SchemaDefinition.AllColumns.ForEach(column =>
    {
        Console.WriteLine($"{column.Name, -40}\\t{column.Type, -20}");
    });
}

```

```

        uiMethods.DisplayTitle("Cluster keys");
        resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
        {

Console.WriteLine($"{clusterKey.Name, -40}\t{clusterKey.OrderBy, -20}");
        });

        uiMethods.DisplayTitle("Partition keys");
        resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
        {
            Console.WriteLine($"{partitionKey.Name}");
        });

        uiMethods.PressEnter();
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }

    // Access Apache Cassandra using the Cassandra drive for C#.
    var cassandraWrapper = host.Services.GetRequiredService<CassandraWrapper>();
    var movieFilePath = configuration["MovieFile"];

    Console.WriteLine("Let's add some movies to the table we created.");
    var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

    uiMethods.PressEnter();

    Console.WriteLine("Added the following movies to the table:");
    var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
    uiMethods.DisplayTitle("All Movies");

    foreach (var row in rows)
    {
        var title = row.GetValue<string>("title");
        var year = row.GetValue<int>("year");
        var plot = row.GetValue<string>("plot");
        var release_date = row.GetValue<DateTime>("release_date");
        Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
        Console.WriteLine(uiMethods.SepBar);
    }

```



```
// Update the table schema
uiMethods.DisplayTitle("Update table schema");
Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

// First save the current time as a UTC Date so the original
// table can be restored later.
var timeChanged = DateTime.UtcNow;

// Now update the schema.
var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
uiMethods.PressEnter();

Console.WriteLine("Now let's mark some of the movies as watched.");

// Pick some files to mark as watched.
var movieToWatch = rows[2].GetValue<string>("title");
var watchedMovieYear = rows[2].GetValue<int>("year");
var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[6].GetValue<string>("title");
watchedMovieYear = rows[6].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[9].GetValue<string>("title");
watchedMovieYear = rows[9].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[10].GetValue<string>("title");
watchedMovieYear = rows[10].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[13].GetValue<string>("title");
watchedMovieYear = rows[13].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

uiMethods.DisplayTitle("Watched movies");
Console.WriteLine("These movies have been marked as watched:");
```

```

        rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            Console.WriteLine($"{title,-40}\t{year,8}");
        }
        uiMethods.PressEnter();

        Console.WriteLine("We can restore the table to its previous state but that
can take up to 20 minutes to complete.");
        string answer;
        do
        {
            Console.WriteLine("Do you want to restore the table? (y/n)");
            answer = Console.ReadLine();
        } while (answer.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            var restoredTableName = $"{tableName}_restored";
            var restoredTableArn = await keyspacesWrapper.RestoreTable(
                keyspaceName,
                tableName,
                restoredTableName,
                timeChanged);
            // Loop and call GetTable until the table is gone. Once it has been
            // deleted completely, GetTable will raise a ResourceNotFoundException.
            bool wasRestored = false;

            try
            {
                do
                {
                    var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                    wasRestored = (resp.Status == TableStatus.ACTIVE);
                } while (!wasRestored);
            }
            catch (ResourceNotFoundException)
            {
                // If the restored table raised an error, it isn't
                // ready yet.
                Console.Write(".");
            }
        }
    
```

```

    }
}

uiMethods.DisplayTitle("Clean up resources.");

// Delete the table.
success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
Console.WriteLine("Waiting for the table to be removed completely. ");

// Loop and call GetTable until the table is gone. Once it has been
// deleted completely, GetTable will raise a ResourceNotFoundException.
bool wasDeleted = false;

try
{
    do
    {
        var resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
    } while (!wasDeleted);
}
catch (ResourceNotFoundException ex)
{
    wasDeleted = true;
    Console.WriteLine($"{ex.Message} indicates that the table has been
deleted.");
}

// Delete the keyspace.
success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
Console.WriteLine("The keyspace has been deleted and the demo is now
complete.");
}
}

```

```
namespace KeyspacesActions;
```

```

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.

```

```
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }

    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
        }
    }
}
```

```

        PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
    };

    var response = await _amazonKeyspaces.CreateTableAsync(request);
    return response.ResourceArn;
}

/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTable(string keyspaceName, string tableName)
{
    var response = await _amazonKeyspaces.DeleteTableAsync(
        new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{

```

```

        var response = await _amazonKeyspaces.GetKeySpaceAsync(
            new GetKeySpaceRequest { KeySpaceName = keySpaceName });
        return response.ResourceArn;
    }

    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keySpaceName">The keySpace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keySpaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeySpaceName = keySpaceName, TableName =
tableName });
        return response;
    }

    /// <summary>
    /// Lists all keySpaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeySpaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeySpaces(new
ListKeySpacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "KeySpace name", "KeySpace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keySpace in paginator.KeySpaces)
        {
            Console.WriteLine($"{keySpace.KeySpaceName, -30}\t{keySpace.ResourceArn}");
        }
    }

    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keySpace.
    /// </summary>

```

```

    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
        { KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }

    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }

    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>

```

```

    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }
}

```

```

using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
/// database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/repository/
sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

```



```

// User name and password for the service.
private string _userName = null!;
private string _pwd = null!;

public CassandraWrapper()
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load test settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

    _localPathToFile = Path.GetTempPath();

    // Get the Starfield digital certificate and save it locally.
    var client = new WebClient();
    client.DownloadFile(_certLocation, $"{_localPathToFile}/{_certFileName}");

    //var httpClient = new HttpClient();
    //var httpResult = httpClient.Get(fileUrl);
    //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
    //using var fileStream = File.Create(pathToSave);
    //resultStream.CopyTo(fileStream);

    _certCollection = new X509Certificate2Collection();
    _amazoncert = new X509Certificate2($"{_localPathToFile}/{_certFileName}");

    // Get the user name and password stored in the configuration file.
    _userName = _configuration["UserName"]!;
    _pwd = _configuration["Password"]!;

    // For a list of Service Endpoints for Amazon Keyspaces, see:
    // https://docs.aws.amazon.com/keyspaces/latest/devguide/
    programmatic.endpoints.html
    var awsEndpoint = _configuration["ServiceEndpoint"];

    _cluster = Cluster.Builder()
        .AddContactPoints(awsEndpoint)
        .WithPort(9142)
        .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
        .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
        .WithQueryOptions(
            new QueryOptions()

```

```

        .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
        .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
    .Build();
}

/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the Apache Cassandra table.
/// </summary>
/// <param name="movieFileName">The full path to the JSON file.</param>
/// <returns>A list of movie objects.</returns>
public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport =
0)
{
    if (!File.Exists(movieFileName))
    {
        return null!;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();

    var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

    // If numToImport = 0, return all movies in the collection.
    if (numToImport == 0)
    {
        // Now return the entire list of movies.
        return allMovies;
    }
    else
    {
        // Now return the first numToImport entries.
        return allMovies.GetRange(0, numToImport);
    }
}

/// <summary>
/// Insert movies into the movie table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="movieTableName">The Amazon Keyspaces table.</param>
/// <param name="movieFilePath">The path to the resource file containing
/// movie data to insert into the table.</param>

```

```

    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
    {
        // Get some movie data from the movies.json file
        var movies = ImportMoviesFromJson(movieFilePath, numToImport);

        var session = _cluster.Connect(keyspaceName);

        string insertCql;

        RowSet rs;

        // Now we insert the numToImport movies into the table.
        foreach (var movie in movies)
        {
            // Escape single quote characters in the plot.
            insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values($${movie.Title}$$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}', $${movie.Info.Plot}$$)";
            rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
        }

        return true;
    }

    /// <summary>
    /// Gets all of the movies in the movies table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing movie data.</returns>
    public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
    }

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}

/// <summary>
/// Mark a movie in the movie table as watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <param name="title">The title of the movie to mark as watched.</param>
/// <param name="year">The year the movie was released.</param>
/// <returns>A set of rows containing the changed data.</returns>
public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
{
    var session = _cluster.Connect();
    string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
WHERE title = ${title} AND year = {year}";
    var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
    var rows = rs.GetRows().ToList();
    return rows;
}

/// <summary>
/// Retrieve the movies in the movies table where watched is true.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <returns>A list of row objects containing information about movies
/// where watched is true.</returns>
public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
tableName)
{
    var session = _cluster.Connect();
    RowSet rs;
    try
    {
        rs = await session.ExecuteAsync(new SimpleStatement($"SELECT title,
year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW FILTERING"));
    }
}

```

```
        // Extract the row data from the returned RowSet.
        var rows = rs.GetRows().ToList();
        return rows;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateKeyspace](#)
 - [CreateTable](#)
 - [DeleteKeyspace](#)
 - [DeleteTable](#)
 - [GetKeyspace](#)
 - [GetTable](#)
 - [ListKeyspaces](#)
 - [ListTables](#)
 - [RestoreTable](#)
 - [UpdateTable](#)

Actions

CreateKeyspace

L'exemple de code suivant montre comment utiliser `CreateKeyspace`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
    var response =
        await _amazonKeyspaces.CreateKeyspaceAsync(
            new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateKeyspace](#) à la section Référence des AWS SDK pour .NET API.

CreateTable

L'exemple de code suivant montre comment utiliser `CreateTable`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a new Amazon Keyspaces table.
```

```
/// </summary>
/// <param name="keyspaceName">The keyspace where the table will be created.</param>
/// <param name="schema">The schema for the new table.</param>
/// <param name="tableName">The name of the new table.</param>
/// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
{
    var request = new CreateTableRequest
    {
        KeyspaceName = keyspaceName,
        SchemaDefinition = schema,
        TableName = tableName,
        PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
    };

    var response = await _amazonKeyspaces.CreateTableAsync(request);
    return response.ResourceArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateTable](#) à la section Référence des AWS SDK pour .NET API.

DeleteKeyspace

L'exemple de code suivant montre comment utiliser `DeleteKeyspace`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an existing keyspace.
```

```

/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteKeyspace](#) à la section Référence des AWS SDK pour .NET API.

DeleteTable

L'exemple de code suivant montre comment utiliser `DeleteTable`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTable(string keyspaceName, string tableName)
{
    var response = await _amazonKeyspaces.DeleteTableAsync(
        new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

```


- Pour plus de détails sur l'API, reportez-vous [DeleteTable](#) à la section Référence des AWS SDK pour .NET API.

GetKeyspace

L'exemple de code suivant montre comment utiliser `GetKeyspace`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.GetKeyspaceAsync(
        new GetKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetKeyspace](#) à la section Référence des AWS SDK pour .NET API.

GetTable

L'exemple de code suivant montre comment utiliser `GetTable`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information about an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the Amazon Keyspaces table.</param>
/// <returns>The response containing data about the table.</returns>
public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
{
    var response = await _amazonKeyspaces.GetTableAsync(
        new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetTable](#) à la section Référence des AWS SDK pour .NET API.

ListKeyspaces

L'exemple de code suivant montre comment utiliser `ListKeyspaces`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Lists all keyspaces for the account.
/// </summary>
/// <returns>Async task.</returns>
public async Task ListKeyspaces()
{
    var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

    Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
    Console.WriteLine(new string('-', Console.WindowWidth));
    await foreach (var keyspace in paginator.Keyspaces)
    {
        Console.WriteLine($"{keyspace.KeyspaceName, -30}\t{keyspace.ResourceArn}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListKeyspaces](#) à la section Référence des AWS SDK pour .NET API.

ListTables

L'exemple de code suivant montre comment utiliser `ListTables`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Lists the Amazon Keyspaces tables in a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>A list of TableSummary objects.</returns>
public async Task<List<TableSummary>> ListTables(string keyspaceName)

```

```

{
    var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
    { KeyspaceName = keyspaceName });
    response.Tables.ForEach(table =>
    {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
    });

    return response.Tables;
}

```

- Pour plus de détails sur l'API, reportez-vous [ListTables](#) à la section Référence des AWS SDK pour .NET API.

RestoreTable

L'exemple de code suivant montre comment utiliser `RestoreTable`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Restores the specified table to the specified point in time.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to restore.</param>
/// <param name="timestamp">The time to which the table will be restored.</
param>
/// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
public async Task<string> RestoreTable(string keyspaceName, string tableName,
string restoredTableName, DateTime timestamp)
{
    var request = new RestoreTableRequest

```

```

    {
        RestoreTimestamp = timestamp,
        SourceKeyspaceName = keyspaceName,
        SourceTableName = tableName,
        TargetKeyspaceName = keyspaceName,
        TargetTableName = restoredTableName
    };

    var response = await _amazonKeyspaces.RestoreTableAsync(request);
    return response.RestoredTableARN;
}

```

- Pour plus de détails sur l'API, reportez-vous [RestoreTable](#) à la section Référence des AWS SDK pour .NET API.

UpdateTable

L'exemple de code suivant montre comment utiliser `UpdateTable`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
        KeyspaceName = keyspaceName,

```

```
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [UpdateTable](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation de Kinesis SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide de Winesis.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Exemples sans serveur](#)

Actions

AddTagsToStream

L'exemple de code suivant montre comment utiliser `AddTagsToStream`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        var tags = new Dictionary<string, string>
        {
            { "Project", "Sample Kinesis Project" },
            { "Application", "Sample Kinesis App" },
        };

        var success = await ApplyTagsToStreamAsync(client, streamName, tags);

        if (success)
        {
            Console.WriteLine($"Taggs successfully added to {streamName}.");
        }
        else
        {
            Console.WriteLine("Tags were not added to the stream.");
        }
    }
}
```

```

    /// <summary>
    /// Applies the set of tags to the named Kinesis stream.
    /// </summary>
    /// <param name="client">The initialized Kinesis client.</param>
    /// <param name="streamName">The name of the Kinesis stream to which
    /// the tags will be attached.</param>
    /// <param name="tags">A dictionary containing key/value pairs which
    /// will be used to create the Kinesis tags.</param>
    /// <returns>A Boolean value which represents the success or failure
    /// of AddTagsToStreamAsync.</returns>
    public static async Task<bool> ApplyTagsToStreamAsync(
        IAmazonKinesis client,
        string streamName,
        Dictionary<string, string> tags)
    {
        var request = new AddTagsToStreamRequest
        {
            StreamName = streamName,
            Tags = tags,
        };

        var response = await client.AddTagsToStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [AddTagsToStream](#) à la section Référence des AWS SDK pour .NET API.

CreateStream

L'exemple de code suivant montre comment utiliser `CreateStream`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to create a new Amazon Kinesis stream.
/// </summary>
public class CreateStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        int shardCount = 1;

        var success = await CreateNewStreamAsync(client, streamName,
shardCount);
        if (success)
        {
            Console.WriteLine($"The stream, {streamName} successfully
created.");
        }
    }

    /// <summary>
    /// Creates a new Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client.</param>
    /// <param name="streamName">The name for the new stream.</param>
    /// <param name="shardCount">The number of shards the new stream will
    /// use. The throughput of the stream is a function of the number of
    /// shards; more shards are required for greater provisioned
```

```
    /// throughput.</param>
    /// <returns>A Boolean value indicating whether the stream was created.</returns>
    public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis client,
        string streamName, int shardCount)
    {
        var request = new CreateStreamRequest
        {
            StreamName = streamName,
            ShardCount = shardCount,
        };

        var response = await client.CreateStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateStream](#) à la section Référence des AWS SDK pour .NET API.

DeleteStream

L'exemple de code suivant montre comment utiliser `DeleteStream`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
```

```

    /// Shows how to delete an Amazon Kinesis stream.
    /// </summary>
    public class DeleteStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string streamName = "AmazonKinesisStream";

            var success = await DeleteStreamAsync(client, streamName);

            if (success)
            {
                Console.WriteLine($"Stream, {streamName} successfully deleted.");
            }
            else
            {
                Console.WriteLine("Stream not deleted.");
            }
        }

        /// <summary>
        /// Deletes a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the string to delete.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }

```

```
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteStream](#) à la section Référence des AWS SDK pour .NET API.

DeregisterStreamConsumer

L'exemple de code suivant montre comment utiliser `DeregisterStreamConsumer`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Kinesis;  
using Amazon.Kinesis.Model;  
  
/// <summary>  
/// Shows how to deregister a consumer from an Amazon Kinesis stream.  
/// </summary>  
public class DeregisterConsumer  
{  
    public static async Task Main(string[] args)  
    {  
        IAmazonKinesis client = new AmazonKinesisClient();  
  
        string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/  
AmazonKinesisStream";  
        string consumerName = "CONSUMER_NAME";  
        string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/  
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";
```

```

        var success = await DeregisterConsumerAsync(client, streamARN,
consumerARN, consumerName);

        if (success)
        {
            Console.WriteLine($"{consumerName} successfully deregistered.");
        }
        else
        {
            Console.WriteLine($"{consumerName} was not successfully
deregistered.");
        }
    }

    /// <summary>
    /// Deregisters a consumer from a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamARN">The ARN of a Kinesis stream.</param>
    /// <param name="consumerARN">The ARN of the consumer.</param>
    /// <param name="consumerName">The name of the consumer.</param>
    /// <returns>A Boolean value representing the success of the operation.</
returns>
    public static async Task<bool> DeregisterConsumerAsync(
        IAmazonKinesis client,
        string streamARN,
        string consumerARN,
        string consumerName)
    {
        var request = new DeregisterStreamConsumerRequest
        {
            StreamARN = streamARN,
            ConsumerARN = consumerARN,
            ConsumerName = consumerName,
        };

        var response = await client.DeregisterStreamConsumerAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeregisterStreamConsumer](#) à la section Référence des AWS SDK pour .NET API.

ListStreamConsumers

L'exemple de code suivant montre comment utiliser `ListStreamConsumers`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// List the consumers of an Amazon Kinesis stream.
/// </summary>
public class ListConsumers
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
        int maxResults = 10;

        var consumers = await ListConsumersAsync(client, streamARN, maxResults);

        if (consumers.Count > 0)
        {
            consumers
                .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
{c.ConsumerARN}"));
        }
    }
}
```

```

        else
        {
            Console.WriteLine("No consumers found.");
        }
    }

    /// <summary>
    /// Retrieve a list of the consumers for a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamARN">The ARN of the stream for which we want to
    /// retrieve a list of clients.</param>
    /// <param name="maxResults">The maximum number of results to return.</
param>
    /// <returns>A list of Consumer objects.</returns>
    public static async Task<List<Consumer>> ListConsumersAsync(IAmazonKinesis
client, string streamARN, int maxResults)
    {
        var request = new ListStreamConsumersRequest
        {
            StreamARN = streamARN,
            MaxResults = maxResults,
        };

        var response = await client.ListStreamConsumersAsync(request);

        return response.Consumers;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListStreamConsumers](#) à la section Référence des AWS SDK pour .NET API.

ListStreams

L'exemple de code suivant montre comment utiliser `ListStreams`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Retrieves and displays a list of existing Amazon Kinesis streams.
/// </summary>
public class ListStreams
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        var response = await client.ListStreamsAsync(new ListStreamsRequest());

        List<string> streamNames = response.StreamNames;

        if (streamNames.Count > 0)
        {
            streamNames
                .ForEach(s => Console.WriteLine($"Stream name: {s}"));
        }
        else
        {
            Console.WriteLine("No streams were found.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListStreams](#) à la section Référence des AWS SDK pour .NET API.

ListTagsForStream

L'exemple de code suivant montre comment utiliser `ListTagsForStream`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to list the tags that have been attached to an Amazon Kinesis
/// stream.
/// </summary>
public class ListTags
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        await ListTagsAsync(client, streamName);
    }

    /// <summary>
    /// List the tags attached to a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the Kinesis stream for which you
    /// wish to display tags.</param>
    public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
    {
        var request = new ListTagsForStreamRequest
        {
```

```

        StreamName = streamName,
        Limit = 10,
    };

    var response = await client.ListTagsForStreamAsync(request);
    DisplayTags(response.Tags);

    while (response.HasMoreTags)
    {
        request.ExclusiveStartTagKey = response.Tags[response.Tags.Count -
1].Key;
        response = await client.ListTagsForStreamAsync(request);
    }
}

/// <summary>
/// Displays the items in a list of Kinesis tags.
/// </summary>
/// <param name="tags">A list of the Tag objects to be displayed.</param>
public static void DisplayTags(List<Tag> tags)
{
    tags
        .ForEach(t => Console.WriteLine($"Key: {t.Key} Value: {t.Value}"));
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ListTagsForStream](#) à la section Référence des AWS SDK pour .NET API.

RegisterStreamConsumer

L'exemple de code suivant montre comment utiliser `RegisterStreamConsumer`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to register a consumer to an Amazon Kinesis
/// stream.
/// </summary>
public class RegisterConsumer
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string consumerName = "NEW_CONSUMER_NAME";
        string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

        var consumer = await RegisterConsumerAsync(client, consumerName,
streamARN);

        if (consumer is not null)
        {
            Console.WriteLine($"{consumer.ConsumerName}");
        }
    }

    /// <summary>
    /// Registers the consumer to a Kinesis stream.
    /// </summary>
    /// <param name="client">The initialized Kinesis client object.</param>
    /// <param name="consumerName">A string representing the consumer.</param>
    /// <param name="streamARN">The ARN of the stream.</param>
    /// <returns>A Consumer object that contains information about the
consumer.</returns>
    public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
client, string consumerName, string streamARN)
    {
        var request = new RegisterStreamConsumerRequest
        {
            ConsumerName = consumerName,
            StreamARN = streamARN,
        };
    }
}

```

```
        var response = await client.RegisterStreamConsumerAsync(request);  
        return response.Consumer;  
    }  
}
```

- Pour plus de détails sur l'API, reportez-vous [RegisterStreamConsumer](#) à la section Référence des AWS SDK pour .NET API.

Exemples sans serveur

Invoquer une fonction Lambda à partir d'un déclencheur Kinesis

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux Kinesis. La fonction récupère la charge utile Kinesis, décode à partir de Base64 et enregistre le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Kinesis avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
using System.Text;  
using Amazon.Lambda.Core;  
using Amazon.Lambda.KinesisEvents;  
using AWS.Lambda.Powertools.Logging;  
  
// Assembly attribute to enable the Lambda function's JSON input to be converted  
// into a .NET class.  
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))  
]
```

```
namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

```
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux Kinesis. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
```

```

    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                }
                };
            }
            Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
            return new StreamsEventResponse();
        }

        private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)

```

```
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

AWS KMS exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with AWS KMS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

CreateAlias

L'exemple de code suivant montre comment utiliser `CreateAlias`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
/// Creates an alias for an AWS Key Management Service (AWS KMS) key.
/// </summary>
public class CreateAlias
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The alias name must start with alias/ and can be
        // up to 256 alphanumeric characters long.
        var aliasName = "alias/ExampleAlias";

        // The value supplied as the TargetKeyId can be either
        // the key ID or key Amazon Resource Name (ARN) of the
        // AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new CreateAliasRequest
        {
            AliasName = aliasName,
            TargetKeyId = keyId,
        };

        var response = await client.CreateAliasAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Alias, {aliasName}, successfully created.");
        }
    }
}
```

```
    }  
    else  
    {  
        Console.WriteLine($"Could not create alias.");  
    }  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateAlias](#) à la section Référence des AWS SDK pour .NET API.

CreateGrant

L'exemple de code suivant montre comment utiliser `CreateGrant`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
public static async Task Main()  
{  
    var client = new AmazonKeyManagementServiceClient();  
  
    // The identity that is given permission to perform the operations  
    // specified in the grant.  
    var grantee = "arn:aws:iam::111122223333:role/ExampleRole";  
  
    // The identifier of the AWS KMS key to which the grant applies. You  
    // can use the key ID or the Amazon Resource Name (ARN) of the KMS key.  
    var keyId = "7c9ecccc2-38cb-4c4f-9db3-766ee8dd3ad4";  
  
    var request = new CreateGrantRequest  
    {  
        GranteePrincipal = grantee,  
        KeyId = keyId,  
    }  
}
```

```
        // A list of operations that the grant allows.
        Operations = new List<string>
        {
            "Encrypt",
            "Decrypt",
        },
    };

    var response = await client.CreateGrantAsync(request);

    string grantId = response.GrantId; // The unique identifier of the
grant.
    string grantToken = response.GrantToken; // The grant token.

    Console.WriteLine($"Id: {grantId}, Token: {grantToken}");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateGrant](#) à la section Référence des AWS SDK pour .NET API.

CreateKey

L'exemple de code suivant montre comment utiliser `CreateKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;
```

```

    /// <summary>
    /// Shows how to create a new AWS Key Management Service (AWS KMS)
    /// key.
    /// </summary>
    public class CreateKey
    {
        public static async Task Main()
        {
            // Note that if you need to create a Key in an AWS Region
            // other than the Region defined for the default user, you need to
            // pass the Region to the client constructor.
            var client = new AmazonKeyManagementServiceClient();

            // The call to CreateKeyAsync will create a symmetrical AWS KMS
            // key. For more information about symmetrical and asymmetrical
            // keys, see:
            //
            // https://docs.aws.amazon.com/kms/latest/developerguide/symm-asymm-
choose.html
            var response = await client.CreateKeyAsync(new CreateKeyRequest());

            // The KeyMetadata object contains information about the new AWS KMS
key.
            KeyMetadata keyMetadata = response.KeyMetadata;

            if (keyMetadata is not null)
            {
                Console.WriteLine($"KMS Key: {keyMetadata.KeyId} was successfully
created.");
            }
            else
            {
                Console.WriteLine("Could not create KMS Key.");
            }
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateKey](#) à la section Référence des AWS SDK pour .NET API.

DescribeKey

L'exemple de code suivant montre comment utiliser `DescribeKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
/// Retrieve information about an AWS Key Management Service (AWS KMS) key.
/// You can supply either the key Id or the key Amazon Resource Name (ARN)
/// to the DescribeKeyRequest KeyId property.
/// </summary>
public class DescribeKey
{
    public static async Task Main()
    {
        var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
        var request = new DescribeKeyRequest
        {
            KeyId = keyId,
        };

        var client = new AmazonKeyManagementServiceClient();

        var response = await client.DescribeKeyAsync(request);
        var metadata = response.KeyMetadata;

        Console.WriteLine($"{metadata.KeyId} created on:
{metadata.CreationDate}");
        Console.WriteLine($"State: {metadata.KeyState}");
        Console.WriteLine($"{metadata.Description}");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeKey](#) à la section Référence des AWS SDK pour .NET API.

DisableKey

L'exemple de code suivant montre comment utiliser `DisableKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
/// Disable an AWS Key Management Service (AWS KMS) key and then retrieve
/// the key's status to show that it has been disabled.
/// </summary>
public class DisableKey
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The identifier of the AWS KMS key to disable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new DisableKeyRequest
        {
            KeyId = keyId,
        };
    }
}
```

```

var response = await client.DisableKeyAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    // Retrieve information about the key to show that it has now
    // been disabled.
    var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
    {
        KeyId = keyId,
    });
    Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DisableKey](#) à la section Référence des AWS SDK pour .NET API.

EnableKey

L'exemple de code suivant montre comment utiliser `EnableKey`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>

```

```

/// Enable an AWS Key Management Service (AWS KMS) key.
/// </summary>
public class EnableKey
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The identifier of the AWS KMS key to enable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new EnableKeyRequest
        {
            KeyId = keyId,
        };

        var response = await client.EnableKeyAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            // Retrieve information about the key to show that it has now
            // been enabled.
            var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
            {
                KeyId = keyId,
            });
            Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [EnableKey](#) à la section Référence des AWS SDK pour .NET API.

ListAliases

L'exemple de code suivant montre comment utiliser `ListAliases`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
/// List the AWS Key Management Service (AWS KMS) aliases that have been defined
for
/// the keys in the same AWS Region as the default user. If you want to list
/// the aliases in a different Region, pass the Region to the client
/// constructor.
/// </summary>
public class ListAliases
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListAliasesRequest();
        var response = new ListAliasesResponse();

        do
        {
            response = await client.ListAliasesAsync(request);

            response.Aliases.ForEach(alias =>
            {
                Console.WriteLine($"Created: {alias.CreationDate} Last Update:
{alias.LastUpdatedDate} Name: {alias.AliasName}");
            });

            request.Marker = response.NextMarker;
        }
        while (response.Truncated);
    }
}
```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [ListAliases](#) à la section Référence des AWS SDK pour .NET API.

ListGrants

L'exemple de code suivant montre comment utiliser `ListGrants`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
/// List the AWS Key Management Service (AWS KMS) grants that are associated
with
/// a specific key.
/// </summary>
public class ListGrants
{
    public static async Task Main()
    {
        // The identifier of the AWS KMS key to disable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListGrantsRequest
        {
            KeyId = keyId,
        };
    }
}
```

```
var response = new ListGrantsResponse();

do
{
    response = await client.ListGrantsAsync(request);

    response.Grants.ForEach(grant =>
    {
        Console.WriteLine($"{grant.GrantId}");
    });

    request.Marker = response.NextMarker;
}
while (response.Truncated);
}
```

- Pour plus de détails sur l'API, reportez-vous [ListGrants](#) à la section Référence des AWS SDK pour .NET API.

ListKeys

L'exemple de code suivant montre comment utiliser `ListKeys`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

/// <summary>
```

```
/// List the AWS Key Managements Service (AWS KMS) keys for the AWS Region
/// of the default user. To list keys in another AWS Region, supply the Region
/// as a parameter to the client constructor.
/// </summary>
public class ListKeys
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListKeysRequest();
        var response = new ListKeysResponse();

        do
        {
            response = await client.ListKeysAsync(request);

            response.Keys.ForEach(key =>
            {
                Console.WriteLine($"ID: {key.KeyId}, {key.KeyArn}");
            });

            // Set the Marker property when response.Truncated is true
            // in order to get the next keys.
            request.Marker = response.NextMarker;
        }
        while (response.Truncated);
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListKeys](#) à la section Référence des AWS SDK pour .NET API.

Exemples Lambda utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide de Lambda.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

AWS les contributions communautaires sont des exemples qui ont été créés et sont maintenus par plusieurs équipes AWS. Pour faire un commentaire, utilisez le mécanisme fourni dans les référentiels liés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)
- [AWS contributions communautaires](#)

Mise en route

Hello Lambda

L'exemple de code suivant montre comment commencer à utiliser Lambda.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace LambdaActions;
```

```
using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your  
existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {
            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListFunctions](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créer un rôle IAM et une fonction Lambda, puis charger le code du gestionnaire.
- Invoquer la fonction avec un seul paramètre et obtenir des résultats.
- Mettre à jour le code de la fonction et configurer avec une variable d'environnement.
- Invoquer la fonction avec de nouveaux paramètres et obtenir des résultats. Afficher le journal d'exécution renvoyé.
- Répertorier les fonctions pour votre compte, puis nettoyer les ressources.

Pour plus d'informations, consultez [Créer une fonction Lambda à l'aide de la console](#).

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez des méthodes qui exécutent des actions Lambda.

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
```

```

public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //             the source code is stored.
    // S3Key      - The name of the file containing the code.
    var functionCode = new FunctionCode
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };

    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}

/// <summary>
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };
}

```



```

    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}

/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}

/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {

```

```

        FunctionName = functionName,
        Payload = payload,
    };

    var response = await _lambdaService.InvokeAsync(request);
    MemoryStream stream = response.Payload;
    string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
    return returnValue;
}

/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}

/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{

```

```

        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
    }

    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
actions.</param>
    /// <param name="environmentVariables">A dictionary of environment variables.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

```
}
```

Créer une fonction qui exécute le scénario.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonLambda>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<LambdaWrapper>()
                    .AddTransient<LambdaRoleWrapper>())
```

```

        .AddTransient<UIWrapper>()
    )
    .Build();

var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load test settings from .json file.
    .AddJsonFile("settings.local.json",
        true) // Optionally load local settings.
    .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<LambdaBasics>();

var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

string functionName = configuration["FunctionName"]!;
string roleName = configuration["RoleName"]!;
string policyDocument = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Principal\": { " +
    "        \"Service\": \"lambda.amazonaws.com\" " +
    "      }, " +
    "      \"Action\": \"sts:AssumeRole\" " +
    "    } " +
    "  ] " +
    "}";

var incrementHandler = configuration["IncrementHandler"];
var calculatorHandler = configuration["CalculatorHandler"];
var bucketName = configuration["BucketName"];
var incrementKey = configuration["IncrementKey"];
var calculatorKey = configuration["CalculatorKey"];
var policyArn = configuration["PolicyArn"];

uiWrapper.DisplayLambdaBasicsOverview();

```

```
// Create the policy to use with the AWS Lambda functions and then attach
the
// policy to a new role.
var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

Console.WriteLine("Waiting for role to become active.");
uiWrapper.WaitABit(15, "Wait until the role is active before trying to use
it.");

// Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
var success = await lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn,
roleName);
uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to the
role.");

// Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
// (Amazon S3) bucket.
uiWrapper.DisplayTitle("Create Lambda Function");
Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
    functionName,
    bucketName,
    incrementKey,
    roleArn,
    incrementHandler);

Console.WriteLine("Waiting for the new function to be available.");
Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

// Get the Lambda function.
Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
FunctionConfiguration config;
do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.State != State.Active);

Console.WriteLine($"\\nThe function, {functionName} has been created.");
```

```

        Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

        uiWrapper.PressEnter();

        // List the Lambda functions.
        uiWrapper.DisplayTitle("Listing all Lambda functions.");
        var functions = await lambdaWrapper.ListFunctionsAsync();
        DisplayFunctionList(functions);

        uiWrapper.DisplayTitle("Invoke increment function");
        Console.WriteLine("Now that it has been created, invoke the Lambda increment
function.");
        string? value;
        do
        {
            Console.Write("Enter a value to increment: ");
            value = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(value));

        string functionParameters = "{" +
            "\"action\": \"increment\", " +
            "\"x\": \"" + value + "\"" +
            "}";
        var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"{value} + 1 = {answer}.");

        uiWrapper.DisplayTitle("Update function");
        Console.WriteLine("Now update the Lambda function code.");
        await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,

```

```

        new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

    do
    {
        config = await lambdaWrapper.GetFunctionAsync(functionName);
        Console.Write(".");
    }
    while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

    uiWrapper.DisplayTitle("Call updated function");
    Console.WriteLine("Now call the updated function...");

    bool done = false;

    do
    {
        string? opSelected;

        Console.WriteLine("Select the operation to perform:");
        Console.WriteLine("\t1. add");
        Console.WriteLine("\t2. subtract");
        Console.WriteLine("\t3. multiply");
        Console.WriteLine("\t4. divide");
        Console.WriteLine("\t0r enter \"q\" to quit.");
        Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the operation");
        you want to perform: ");
        do
        {
            Console.Write("Your choice? ");
            opSelected = Console.ReadLine();
        }
        while (opSelected == string.Empty);

        var operation = (opSelected) switch
        {
            "1" => "add",
            "2" => "subtract",
            "3" => "multiply",
            "4" => "divide",
            "q" => "quit",
            _ => "add",
        };

        if (operation == "quit")

```



```

        {
            done = true;
        }
        else
        {
            // Get two numbers and an action from the user.
            value = string.Empty;
            do
            {
                Console.Write("Enter the first value: ");
                value = Console.ReadLine();
            }
            while (value == string.Empty);

            string? value2;
            do
            {
                Console.Write("Enter a second value: ");
                value2 = Console.ReadLine();
            }
            while (value2 == string.Empty);

            functionParameters = "{" +
                "\"action\": \"" + operation + "\", " +
                "\"x\": \"" + value + "\", " +
                "\"y\": \"" + value2 + "\"" +
                "}";

            answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
            Console.WriteLine($"The answer when we {operation} the two numbers
is: {answer}.");
        }

        uiWrapper.PressEnter();
    } while (!done);

    // Delete the function created earlier.

    uiWrapper.DisplayTitle("Clean up resources");
    // Detach the IAM policy from the IAM role.
    Console.WriteLine("First detach the IAM policy from the role.");
    success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);

```

```

        uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached from
the role.");

        Console.WriteLine("Delete the AWS Lambda function.");
        success = await lambdaWrapper.DeleteFunctionAsync(functionName);
        if (success)
        {
            Console.WriteLine($"The {functionName} function was deleted.");
        }
        else
        {
            Console.WriteLine($"Could not remove the function {functionName}");
        }

        // Now delete the IAM role created for use with the functions
        // created by the application.
        Console.WriteLine("Now we can delete the role that we created.");
        success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
        if (success)
        {
            Console.WriteLine("The role has been successfully removed.");
        }
        else
        {
            Console.WriteLine("Couldn't delete the role.");
        }

        Console.WriteLine("The Lambda Scenario is now complete.");
        uiWrapper.PressEnter();

        // Displays a formatted list of existing functions returned by the
        // LambdaMethods.ListFunctions.
        void DisplayFunctionList(List<FunctionConfiguration> functions)
        {
            functions.ForEach(functionConfig =>
            {
                Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
            });
        }
    }
}

```

```

namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,

```

```

        RoleName = roleName,
    };

    var response = await _lambdaRoleService.CreateRoleAsync(request);
    return response.Role.Arn;
}

/// <summary>
/// Deletes an IAM role.
/// </summary>
/// <param name="roleName">The name of the role to delete.</param>
/// <returns>A Boolean value indicating the success of the operation.</returns>
public async Task<bool> DeleteLambdaRoleAsync(string roleName)
{
    var request = new DeleteRoleRequest
    {
        RoleName = roleName,
    };

    var response = await _lambdaRoleService.DeleteRoleAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
roleName)
{
    var response = await _lambdaRoleService.DetachRolePolicyAsync(new
DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
}

namespace LambdaScenarioCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {

```

```

        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM)
role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the value
passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>

```

```

    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}

```

Définir un gestionnaire Lambda qui incrémente un nombre.

```

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaIncrement;

```

```

public class Function
{
    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}

```

Définir un deuxième gestionnaire Lambda qui effectue des opérations arithmétiques.

```

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaCalculator;

public class Function
{

```

```
/// <summary>
/// A simple function that takes two number in string format and performs
/// the requested arithmetic function.
/// </summary>
/// <param name="input">JSON data containing an action, and x and y values.
/// Valid actions include: add, subtract, multiply, and divide.</param>
/// <param name="context">The context object passed by Lambda containing
/// information about invocation, function, and execution environment.</param>
/// <returns>A string representing the results of the calculation.</returns>
public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
{
    var action = input["action"];
    int x = Convert.ToInt32(input["x"]);
    int y = Convert.ToInt32(input["y"]);
    int result;
    switch (action)
    {
        case "add":
            result = x + y;
            break;
        case "subtract":
            result = x - y;
            break;
        case "multiply":
            result = x * y;
            break;
        case "divide":
            if (y == 0)
            {
                Console.Error.WriteLine("Divide by zero error.");
                result = 0;
            }
            else
            {
                result = x / y;
            }
            break;
        default:
            Console.Error.WriteLine($"{action} is not a valid operation.");
            result = 0;
            break;
    }
    return result;
}
```


- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Actions

CreateFunction

L'exemple de code suivant montre comment utiliser `CreateFunction`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
```

```
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //           the source code is stored.
    // S3Key     - The name of the file containing the code.
    var functionCode = new FunctionCode
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };

    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateFunction](#) à la section Référence des AWS SDK pour .NET API.

DeleteFunction

L'exemple de code suivant montre comment utiliser `DeleteFunction`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteFunction](#) à la section Référence des AWS SDK pour .NET API.

GetFunction

L'exemple de code suivant montre comment utiliser `GetFunction`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetFunction](#) à la section Référence des AWS SDK pour .NET API.

Invoke

L'exemple de code suivant montre comment utiliser `Invoke`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {
        FunctionName = functionName,
        Payload = payload,
    };

    var response = await _lambdaService.InvokeAsync(request);
    MemoryStream stream = response.Payload;
    string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
    return returnValue;
}
```

- Pour en savoir plus sur l'API, consultez [Invoke](#) dans la Référence de l'API AWS SDK pour .NET.

ListFunctions

L'exemple de code suivant montre comment utiliser `ListFunctions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListFunctions](#) à la section Référence des AWS SDK pour .NET API.

UpdateFunctionCode

L'exemple de code suivant montre comment utiliser `UpdateFunctionCode`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
```

```
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{
    var functionCodeRequest = new UpdateFunctionCodeRequest
    {
        FunctionName = functionName,
        Publish = true,
        S3Bucket = bucketName,
        S3Key = key,
    };

    var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
    Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
}
```

- Pour plus de détails sur l'API, reportez-vous [UpdateFunctionCode](#) à la section Référence des AWS SDK pour .NET API.

UpdateFunctionConfiguration

L'exemple de code suivant montre comment utiliser `UpdateFunctionConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Update the code of a Lambda function.
```

```
/// </summary>
/// <param name="functionName">The name of the function to update.</param>
/// <param name="functionHandler">The code that performs the function's
actions.</param>
/// <param name="environmentVariables">A dictionary of environment variables.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> UpdateFunctionConfigurationAsync(
    string functionName,
    string functionHandler,
    Dictionary<string, string> environmentVariables)
{
    var request = new UpdateFunctionConfigurationRequest
    {
        Handler = functionHandler,
        FunctionName = functionName,
        Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
    };

    var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

    Console.WriteLine(response.LastModified);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [UpdateFunctionConfiguration](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.

- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Transformation de données avec S3 Object Lambda

L'exemple de code suivant illustre comment transformer des données pour votre application avec S3 Object Lambda.

SDK pour .NET

Explique comment ajouter du code personnalisé aux requêtes GET S3 standard afin de modifier l'objet demandé extrait de S3 pour qu'il réponde aux besoins du client ou de l'application qui le demande.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Lambda
- Amazon S3

Exemples sans serveur

Connexion à une base de données Amazon RDS dans une fonction Lambda

L'exemple de code suivant montre comment implémenter une fonction Lambda qui se connecte à une base de données RDS. La fonction effectue une simple requête de base de données et renvoie le résultat.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de .NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
    /// Handles the Lambda function execution for connecting to RDS using IAM
    authentication.
    /// </summary>
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
    FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
```

```

{
    // Sample Input: {"body": "{\"key1\": \"20\", \"key2\": \"25\"}"}
    var input = JsonSerializer.Deserialize<InputModel>(request.Body);

    /// Obtain authentication token
    var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
        Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
        Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
        Environment.GetEnvironmentVariable("RDS_USERNAME")
    );

    /// Build the Connection String with the Token
    string connectionString =
        $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +

        $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

        $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
        $"Pwd={authToken}";

    try
    {
        await using var connection = new MySqlConnection(connectionString);
        await connection.OpenAsync();

        const string sql = "SELECT @param1 + @param2 AS Sum";

        await using var command = new MySqlCommand(sql, connection);
        command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
"0"));
        command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
"0"));

        await using var reader = await command.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            int result = reader.GetInt32("Sum");

            //Sample Response: {"statusCode":200,"body":{"\"message\": \"The sum
is: 45\"},"isBase64Encoded":false}
            return new APIGatewayProxyResponse
            {
                StatusCode = 200,

```

```
        Body = JsonSerializer.Serialize(new { message = $"The sum is:
{result}" });
    };
}

}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}

return new APIGatewayProxyResponse
{
    StatusCode = 500,
    Body = JsonSerializer.Serialize(new { error = "Internal server error" })
};
}
}
```

Invoker une fonction Lambda à partir d'un déclencheur Kinesis

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux Kinesis. La fonction récupère la charge utile Kinesis, décode à partir de Base64 et enregistre le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Kinesis avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
```

```

using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    }
}

```

```
private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}
```

Invocation d'une fonction Lambda à partir d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux DynamoDB. La fonction récupère les données utiles DynamoDB et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement DynamoDB avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
```

```
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
{dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon DocumentDB

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un flux de modifications DocumentDB. La fonction récupère les données utiles DocumentDB et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Amazon DocumentDB avec Lambda en utilisant .NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
```



```
//Assembly attribute to enable the Lambda function's JSON input to be converted into
//a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaDocDb;

public class Function
{
    /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

    private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
    ILambdaContext context)
    {
        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument, new
        JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }
}
```

```
public class Event
{
    [JsonPropertyName("eventSourceArn")]
    public string EventSourceArn { get; set; }

    [JsonPropertyName("events")]
    public List<DocumentDBEventRecord> Events { get; set; }

    [JsonPropertyName("eventSource")]
    public string EventSource { get; set; }
}

public class DocumentDBEventRecord
{
    [JsonPropertyName("event")]
    public EventData Event { get; set; }
}

public class EventData
{
    [JsonPropertyName("_id")]
    public IdData Id { get; set; }

    [JsonPropertyName("clusterTime")]
    public ClusterTime ClusterTime { get; set; }

    [JsonPropertyName("documentKey")]
    public DocumentKey DocumentKey { get; set; }

    [JsonPropertyName("fullDocument")]
    public Dictionary<string, object> FullDocument { get; set; }

    [JsonPropertyName("ns")]
    public Namespace Namespace { get; set; }

    [JsonPropertyName("operationType")]
    public string OperationType { get; set; }
}

public class IdData
{
    [JsonPropertyName("_data")]
```

```
        public string Data { get; set; }
    }

    public class ClusterTime
    {
        [JsonPropertyName("$timestamp")]
        public Timestamp Timestamp { get; set; }
    }

    public class Timestamp
    {
        [JsonPropertyName("t")]
        public long T { get; set; }

        [JsonPropertyName("i")]
        public int I { get; set; }
    }

    public class DocumentKey
    {
        [JsonPropertyName("_id")]
        public Id Id { get; set; }
    }

    public class Id
    {
        [JsonPropertyName("$oid")]
        public string Oid { get; set; }
    }

    public class Namespace
    {
        [JsonPropertyName("db")]
        public string Db { get; set; }

        [JsonPropertyName("coll")]
        public string Coll { get; set; }
    }
}
```

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon MSK

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un cluster Amazon MSK. La fonction récupère les données utiles MSK et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Amazon MSK avec Lambda en utilisant .NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace MSKLambda;

public class Function
{
    /// <param name="input">The event for the Lambda function handler to process.</param>
    /// <param name="context">The ILambdaContext that provides methods for logging
    and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
```

```
        Console.WriteLine("Key:" + record.Key);
        foreach (var eventRecord in record.Value)
        {
            var valueBytes = eventRecord.Value.ToArray();
            var valueText = Encoding.UTF8.GetString(valueBytes);

            Console.WriteLine("Message:" + valueText);
        }
    }
}
```

Invoquer une fonction lambda à partir d'un déclencheur Amazon S3

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par le chargement d'un objet vers un compartiment S3. La fonction extrait le nom du compartiment S3 et la clé de l'objet à partir du paramètre d'événement et appelle l'API Amazon S3 pour récupérer et consigner le type de contenu de l'objet.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Utilisation d'un événement S3 avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");
            }
        }
    }
}
```

```

        return string.Empty;
    }
}
}
}

```

Invocation d'une fonction lambda à partir d'un déclencheur Amazon SNS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une rubrique SNS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement SNS avec Lambda à l'aide de .NET.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {

```

```
        await ProcessRecordAsync(record, context);
    }
    context.Logger.LogInformation("done");
}

private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext context)
{
    try
    {
        context.Logger.LogInformation($"Processed record {record.Sns.Message}");

        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        //Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Invoyer une fonction Lambda à partir d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une file d'attente SQS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Utilisation d'un événement SQS avec Lambda en utilisant .NET.


```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            //Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Kinesis

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux Kinesis. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots Kinesis avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
```

```

    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                }
                };
            }
            Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
            return new StreamsEventResponse();
        }

        private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)

```

```

    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}

```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur DynamoDB

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'un flux DynamoDB. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots DynamoDB avec Lambda à l'aide de .NET.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;

```

```

using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}

```

```
}
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'une file d'attente SQS. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
```

```

        try
        {
            //process your message
            await ProcessMessageAsync(message, context);
        }
        catch (System.Exception)
        {
            //Add failed message identifier to the batchItemFailures list
            batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
        }
    }
    return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
    ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}

```

AWS contributions communautaires

Création et test d'une application sans serveur

L'exemple de code suivant montre comment créer et tester une application sans serveur à l'aide d'API Gateway avec Lambda et DynamoDB.

SDK pour .NET

Illustre comment créer et tester une application sans serveur composée d'une API Gateway avec Lambda et DynamoDB à l'aide du kit SDK .NET.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda

MediaConvert exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with MediaConvert.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Actions](#)

Mise en route

Bonjour MediaConvert

L'exemple de code suivant montre comment démarrer avec AWS Elemental MediaConvert.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.MediaConvert;
```



```
using Amazon.MediaConvert.Model;

namespace MediaConvertActions;

public static class HelloMediaConvert
{
    static async Task Main(string[] args)
    {
        // Create the client using the default profile.
        var mediaConvertClient = new AmazonMediaConvertClient();

        Console.WriteLine($"Hello AWS Elemental MediaConvert! Your MediaConvert Jobs are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get some MediaConvert jobs.
        var response = await mediaConvertClient.ListJobsAsync(
            new ListJobsRequest()
            {
                MaxResults = 10
            }
        );

        foreach (var job in response.Jobs)
        {
            Console.WriteLine($"\\tJob: {job.Id} status {job.Status}");
            Console.WriteLine();
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeEndpoints](#) à la section Référence des AWS SDK pour .NET API.

Actions

CreateJob

L'exemple de code suivant montre comment utiliser `CreateJob`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Configurez les emplacements des fichiers, le client et l'encapsuleur.

```
// MediaConvert role Amazon Resource Name (ARN).
// For information on creating this role, see
// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
var mediaConvertRole = _configuration["mediaConvertRoleARN"];

// Include the file input and output locations in settings.json or
settings.local.json.
var fileInput = _configuration["fileInput"];
var fileOutput = _configuration["fileOutput"];

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

var wrapper = new MediaConvertWrapper(mcClient);
```

```
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Creating job for input file {fileInput}.");
var jobId = await wrapper.CreateJob(mediaConvertRole!, fileInput!,
fileOutput!);
Console.WriteLine($"Created job with Job ID: {jobId}");
Console.WriteLine(new string('-', 80));
```

Créez la tâche à l'aide de la méthode d'encapsuleur et renvoyez l'ID de la tâche.

```
/// <summary>
/// Create a job to convert a media file.
/// </summary>
```

```

    /// <param name="mediaConvertRole">The Amazon Resource Name (ARN) of the media
    convert role, as specified here:
    /// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-in-
mediaconvert-configured.html</param>
    /// <param name="fileInput">The Amazon Simple Storage Service (Amazon S3)
    location of the input media file.</param>
    /// <param name="fileOutput">The Amazon S3 location for the output media file.</
param>
    /// <returns>The ID of the new job.</returns>
    public async Task<string> CreateJob(string mediaConvertRole, string fileInput,
        string fileOutput)
    {
        CreateJobRequest createJobRequest = new CreateJobRequest
        {
            Role = mediaConvertRole
        };

        createJobRequest.UserMetadata.Add("Customer", "Amazon");

        JobSettings jobSettings = new JobSettings
        {
            AdAvailOffset = 0,
            TimecodeConfig = new TimecodeConfig
            {
                Source = TimecodeSource.EMBEDDED
            }
        };
        createJobRequest.Settings = jobSettings;

        #region OutputGroup

        OutputGroup ofg = new OutputGroup
        {
            Name = "File Group",
            OutputGroupSettings = new OutputGroupSettings
            {
                Type = OutputGroupType.FILE_GROUP_SETTINGS,
                FileGroupSettings = new FileGroupSettings
                {
                    Destination = fileOutput
                }
            }
        };
    }

```

```
Output output = new Output
{
    NameModifier = "_1"
};

#region VideoDescription

VideoDescription vdes = new VideoDescription
{
    ScalingBehavior = ScalingBehavior.DEFAULT,
    TimecodeInsertion = VideoTimecodeInsertion.DISABLED,
    AntiAlias = AntiAlias.ENABLED,
    Sharpness = 50,
    AfdSignaling = AfdSignaling.NONE,
    DropFrameTimecode = DropFrameTimecode.ENABLED,
    RespondToAfd = RespondToAfd.NONE,
    ColorMetadata = ColorMetadata.INSERT,
    CodecSettings = new VideoCodecSettings
    {
        Codec = VideoCodec.H_264
    }
};
output.VideoDescription = vdes;

H264Settings h264 = new H264Settings
{
    InterlaceMode = H264InterlaceMode.PROGRESSIVE,
    NumberReferenceFrames = 3,
    Syntax = H264Syntax.DEFAULT,
    Softness = 0,
    GopClosedCadence = 1,
    GopSize = 90,
    Slices = 1,
    GopBReference = H264GopBReference.DISABLED,
    SlowPal = H264SlowPal.DISABLED,
    SpatialAdaptiveQuantization = H264SpatialAdaptiveQuantization.ENABLED,
    TemporalAdaptiveQuantization = H264TemporalAdaptiveQuantization.ENABLED,
    FlickerAdaptiveQuantization = H264FlickerAdaptiveQuantization.DISABLED,
    EntropyEncoding = H264EntropyEncoding.CABAC,
    Bitrate = 5000000,
    FramerateControl = H264FramerateControl.SPECIFIED,
    RateControlMode = H264RateControlMode.CBR,
    CodecProfile = H264CodecProfile.MAIN,
    Telecine = H264Telecine.NONE,
```

```

        MinIInterval = 0,
        AdaptiveQuantization = H264AdaptiveQuantization.HIGH,
        CodecLevel = H264CodecLevel.AUTO,
        FieldEncoding = H264FieldEncoding.PAFF,
        SceneChangeDetect = H264SceneChangeDetect.ENABLED,
        QualityTuningLevel = H264QualityTuningLevel.SINGLE_PASS,
        FramerateConversionAlgorithm =
            H264FramerateConversionAlgorithm.DUPLICATE_DROP,
        UnregisteredSeiTimecode = H264UnregisteredSeiTimecode.DISABLED,
        GopSizeUnits = H264GopSizeUnits.FRAMES,
        ParControl = H264ParControl.SPECIFIED,
        NumberBFramesBetweenReferenceFrames = 2,
        RepeatPps = H264RepeatPps.DISABLED,
        FramerateNumerator = 30,
        FramerateDenominator = 1,
        ParNumerator = 1,
        ParDenominator = 1
    };
    output.VideoDescription.CodecSettings.H264Settings = h264;

#endregion VideoDescription

#region AudioDescription

AudioDescription ades = new AudioDescription
{
    LanguageCodeControl = AudioLanguageCodeControl.FOLLOW_INPUT,
    // This name matches one specified in the following Inputs.
    AudioSourceName = "Audio Selector 1",
    CodecSettings = new AudioCodecSettings
    {
        Codec = AudioCodec.AAC
    }
};

AacSettings aac = new AacSettings
{
    AudioDescriptionBroadcasterMix =
AacAudioDescriptionBroadcasterMix.NORMAL,
    RateControlMode = AacRateControlMode.CBR,
    CodecProfile = AacCodecProfile.LC,
    CodingMode = AacCodingMode.CODING_MODE_2_0,
    RawFormat = AacRawFormat.NONE,
    SampleRate = 48000,

```

```

        Specification = AacSpecification.MPEG4,
        Bitrate = 64000
    };
    ades.CodecSettings.AacSettings = aac;
    output.AudioDescriptions.Add(ades);

#endregion AudioDescription

#region Mp4 Container

output.ContainerSettings = new ContainerSettings
{
    Container = ContainerType.MP4
};
Mp4Settings mp4 = new Mp4Settings
{
    CslgAtom = Mp4CslgAtom.INCLUDE,
    FreeSpaceBox = Mp4FreeSpaceBox.EXCLUDE,
    MoovPlacement = Mp4MoovPlacement.PROGRESSIVE_DOWNLOAD
};
output.ContainerSettings.Mp4Settings = mp4;

#endregion Mp4 Container

ofg.Outputs.Add(output);
createJobRequest.Settings.OutputGroups.Add(ofg);

#endregion OutputGroup

#region Input

Input input = new Input
{
    FilterEnable = InputFilterEnable.AUTO,
    PsiControl = InputPsiControl.USE_PSI,
    FilterStrength = 0,
    DeblockFilter = InputDeblockFilter.DISABLED,
    DenoiseFilter = InputDenoiseFilter.DISABLED,
    TimecodeSource = InputTimecodeSource.EMBEDDED,
    FileInput = fileInput
};

AudioSelector audsel = new AudioSelector
{

```

```

        Offset = 0,
        DefaultSelection = AudioDefaultSelection.NOT_DEFAULT,
        ProgramSelection = 1,
        SelectorType = AudioSelectorType.TRACK
    };
    audsel.Tracks.Add(1);
    input.AudioSelectors.Add("Audio Selector 1", audsel);

    input.VideoSelector = new VideoSelector
    {
        ColorSpace = ColorSpace.FOLLOW
    };

    createJobRequest.Settings.Inputs.Add(input);

    #endregion Input

    CreateJobResponse createJobResponse =
        await _amazonMediaConvert.CreateJobAsync(createJobRequest);

    var jobId = createJobResponse.Job.Id;

    return jobId;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateJob](#) à la section Référence des AWS SDK pour .NET API.

GetJob

L'exemple de code suivant montre comment utiliser `GetJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Configurez les emplacements des fichiers, le client et l'encapsuleur.

```

        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);

```

Obtenez une tâche grâce à son ID.

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Getting job information for Job ID {jobId}");
        var job = await wrapper.GetJobById(jobId);
        Console.WriteLine($"Job {job.Id} created on {job.CreatedAt:d} has status
{job.Status}.");
        Console.WriteLine(new string('-', 80));

```

```

/// <summary>
/// Get the job information for a job by its ID.
/// </summary>
/// <param name="jobId">The ID of the job.</param>
/// <returns>The Job object.</returns>
public async Task<Job> GetJobById(string jobId)
{
    var jobResponse = await _amazonMediaConvert.GetJobAsync(
        new GetJobRequest
        {
            Id = jobId
        });

    return jobResponse.Job;
}

```


- Pour plus de détails sur l'API, reportez-vous [GetJob](#) à la section Référence des AWS SDK pour .NET API.

ListJobs

L'exemple de code suivant montre comment utiliser `ListJobs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Configurez les emplacements des fichiers, le client et l'encapsuleur.

```
// MediaConvert role Amazon Resource Name (ARN).  
// For information on creating this role, see  
// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-  
in-mediaconvert-configured.html.  
var mediaConvertRole = _configuration["mediaConvertRoleARN"];  
  
// Include the file input and output locations in settings.json or  
settings.local.json.  
var fileInput = _configuration["fileInput"];  
var fileOutput = _configuration["fileOutput"];  
  
AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();  
  
var wrapper = new MediaConvertWrapper(mcClient);
```

Répertoriez les tâches avec un état particulier.

```
Console.WriteLine(new string('-', 80));  
Console.WriteLine($"Listing all complete jobs.");  
var completeJobs = await wrapper.ListAllJobsByStatus(JobStatus.COMPLETE);
```

```
completeJobs.ForEach(j =>
{
    Console.WriteLine($"Job {j.Id} created on {j.CreatedAt:d} has status {j.Status}.");
});
```

Répertoriez les tâches à l'aide d'un programme de pagination.

```
/// <summary>
/// List all of the jobs with a particular status using a paginator.
/// </summary>
/// <param name="status">The status to use when listing jobs.</param>
/// <returns>The list of jobs matching the status.</returns>
public async Task<List<Job>> ListAllJobsByStatus(JobStatus? status = null)
{
    var returnedJobs = new List<Job>();

    var paginatedJobs = _amazonMediaConvert.Paginators.ListJobs(
        new ListJobsRequest
        {
            Status = status
        });

    // Get the entire list using the paginator.
    await foreach (var job in paginatedJobs.Jobs)
    {
        returnedJobs.Add(job);
    }

    return returnedJobs;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListJobs](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon MSK utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon MSK.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Exemples sans serveur](#)

Exemples sans serveur

Invocation d'une fonction Lambda à partir d'un déclencheur Amazon MSK

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception d'enregistrements à partir d'un cluster Amazon MSK. La fonction récupère les données utiles MSK et journalise le contenu de l'enregistrement.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement Amazon MSK avec Lambda en utilisant .NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace MSKLambda;

public class Function
{

```

```
/// <param name="input">The event for the Lambda function handler to process.</  
param>  
/// <param name="context">The ILambdaContext that provides methods for logging  
and describing the Lambda environment.</param>  
/// <returns></returns>  
public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)  
{  
  
    foreach (var record in evnt.Records)  
    {  
        Console.WriteLine("Key:" + record.Key);  
        foreach (var eventRecord in record.Value)  
        {  
            var valueBytes = eventRecord.Value.ToArray();  
            var valueText = Encoding.UTF8.GetString(valueBytes);  
  
            Console.WriteLine("Message:" + valueText);  
        }  
    }  
}
```

Organisations utilisant des exemples SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with Organizations.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

AttachPolicy

L'exemple de code suivant montre comment utiliser `AttachPolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Shows how to attach an AWS Organizations policy to an organization,
/// an organizational unit, or an account.
/// </summary>
public class AttachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then calls the
    /// AttachPolicyAsync method to attach the policy to the root
    /// organization.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var policyId = "p-000000000";
        var targetId = "r-0000";

        var request = new AttachPolicyRequest
        {
            PolicyId = policyId,
            TargetId = targetId,
        };
    }
}
```

```
var response = await client.AttachPolicyAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Successfully attached Policy ID {policyId} to
Target ID: {targetId}.");
}
else
{
    Console.WriteLine("Was not successful in attaching the policy.");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [AttachPolicy](#) à la section Référence des AWS SDK pour .NET API.

CreateAccount

L'exemple de code suivant montre comment utiliser `CreateAccount`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Creates a new AWS Organizations account.
/// </summary>
public class CreateAccount
{

```

```
/// <summary>
/// Initializes an Organizations client object and uses it to create
/// the new account with the name specified in accountName.
/// </summary>
public static async Task Main()
{
    IAmazonOrganizations client = new AmazonOrganizationsClient();
    var accountName = "ExampleAccount";
    var email = "someone@example.com";

    var request = new CreateAccountRequest
    {
        AccountName = accountName,
        Email = email,
    };

    var response = await client.CreateAccountAsync(request);
    var status = response.CreateAccountStatus;

    Console.WriteLine($"The status of {status.AccountName} is
{status.State}.");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateAccount](#) à la section Référence des AWS SDK pour .NET API.

CreateOrganization

L'exemple de code suivant montre comment utiliser `CreateOrganization`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Creates an organization in AWS Organizations.
/// </summary>
public class CreateOrganization
{
    /// <summary>
    /// Creates an Organizations client object and then uses it to create
    /// a new organization with the default user as the administrator, and
    /// then displays information about the new organization.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.CreateOrganizationAsync(new
CreateOrganizationRequest
        {
            FeatureSet = "ALL",
        });

        Organization newOrg = response.Organization;

        Console.WriteLine($"Organization: {newOrg.Id} Main Account:
{newOrg.MasterAccountId}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateOrganization](#) à la section Référence des AWS SDK pour .NET API.

CreateOrganizationalUnit

L'exemple de code suivant montre comment utiliser `CreateOrganizationalUnit`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Creates a new organizational unit in AWS Organizations.
/// </summary>
public class CreateOrganizationalUnit
{
    /// <summary>
    /// Initializes an Organizations client object and then uses it to call
    /// the CreateOrganizationalUnit method. If the call succeeds, it
    /// displays information about the new organizational unit.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitName = "ProductDevelopmentUnit";

        var request = new CreateOrganizationalUnitRequest
        {
            Name = orgUnitName,
            ParentId = "r-0000",
        };

        var response = await client.CreateOrganizationalUnitAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully created organizational unit:
{orgUnitName}.");
        }
    }
}
```

```

        Console.WriteLine($"Organizational unit {orgUnitName} Details");
        Console.WriteLine($"ARN: {response.OrganizationalUnit.Arn} Id:
{response.OrganizationalUnit.Id}");
    }
    else
    {
        Console.WriteLine("Could not create new organizational unit.");
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateOrganizationalUnit](#) à la section Référence des AWS SDK pour .NET API.

CreatePolicy

L'exemple de code suivant montre comment utiliser `CreatePolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Creates a new AWS Organizations Policy.
/// </summary>
public class CreatePolicy
{
    /// <summary>
    /// Initializes the AWS Organizations client object, uses it to
    /// create a new Organizations Policy, and then displays information

```

```

    /// about the newly created Policy.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var policyContent = "{" +
            "  \"Version\": \"2012-10-17\", \" +
            "  \"Statement\" : [{" +
            "    \"Action\" : [\"s3:*\"], \" +
            "    \"Effect\" : \"Allow\", \" +
            "    \"Resource\" : \"*\" \" +
            "  }]" +
            "}";

        try
        {
            var response = await client.CreatePolicyAsync(new
CreatePolicyRequest
            {
                Content = policyContent,
                Description = "Enables admins of attached accounts to delegate
all Amazon S3 permissions",
                Name = "AllowAllS3Actions",
                Type = "SERVICE_CONTROL_POLICY",
            });

            Policy policy = response.Policy;
            Console.WriteLine($"{policy.PolicySummary.Name} has the following
content: {policy.Content}");
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreatePolicy](#) à la section Référence des AWS SDK pour .NET API.

DeleteOrganization

L'exemple de code suivant montre comment utiliser DeleteOrganization.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Shows how to delete an existing organization using the AWS
/// Organizations Service.
/// </summary>
public class DeleteOrganization
{
    /// <summary>
    /// Initializes the Organizations client and then calls
    /// DeleteOrganizationAsync to delete the organization.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.DeleteOrganizationAsync(new
DeleteOrganizationRequest());

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Successfully deleted organization.");
        }
        else
        {
            Console.WriteLine("Could not delete organization.");
        }
    }
}
```

```
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteOrganization](#) à la section Référence des AWS SDK pour .NET API.

DeleteOrganizationalUnit

L'exemple de code suivant montre comment utiliser `DeleteOrganizationalUnit`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Organizations;  
using Amazon.Organizations.Model;  
  
/// <summary>  
/// Shows how to delete an existing AWS Organizations organizational unit.  
/// </summary>  
public class DeleteOrganizationalUnit  
{  
    /// <summary>  
    /// Initializes the Organizations client object and calls  
    /// DeleteOrganizationalUnitAsync to delete the organizational unit  
    /// with the selected ID.  
    /// </summary>  
    public static async Task Main()  
    {  
        // Create the client object using the default account.  
        IAmazonOrganizations client = new AmazonOrganizationsClient();  
  
        var orgUnitId = "ou-0000-000000000";  
    }  
}
```

```
var request = new DeleteOrganizationalUnitRequest
{
    OrganizationalUnitId = orgUnitId,
};

var response = await client.DeleteOrganizationalUnitAsync(request);

if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
{
    Console.WriteLine($"Successfully deleted the organizational unit
with ID: {orgUnitId}.");
}
else
{
    Console.WriteLine($"Could not delete the organizational unit with
ID: {orgUnitId}.");
}
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteOrganizationalUnit](#) à la section Référence des AWS SDK pour .NET API.

DeletePolicy

L'exemple de code suivant montre comment utiliser `DeletePolicy`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
```

```
using Amazon.Organizations.Model;

/// <summary>
/// Deletes an existing AWS Organizations policy.
/// </summary>
public class DeletePolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// delete the policy with the specified policyId.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-000000000";

        var request = new DeletePolicyRequest
        {
            PolicyId = policyId,
        };

        var response = await client.DeletePolicyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted Policy: {policyId}.");
        }
        else
        {
            Console.WriteLine($"Could not delete Policy: {policyId}.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeletePolicy](#) à la section Référence des AWS SDK pour .NET API.

DetachPolicy

L'exemple de code suivant montre comment utiliser DetachPolicy.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Shows how to detach a policy from an AWS Organizations organization,
/// organizational unit, or account.
/// </summary>
public class DetachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and uses it to call
    /// DetachPolicyAsync to detach the policy.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-000000000";
        var targetId = "r-0000";

        var request = new DetachPolicyRequest
        {
            PolicyId = policyId,
            TargetId = targetId,
        };

        var response = await client.DetachPolicyAsync(request);
    }
}
```



```
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully detached policy with Policy Id:
{policyId}.");
        }
        else
        {
            Console.WriteLine("Could not detach the policy.");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DetachPolicy](#) à la section Référence des AWS SDK pour .NET API.

ListAccounts

L'exemple de code suivant montre comment utiliser `ListAccounts`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Uses the AWS Organizations service to list the accounts associated
/// with the default account.
/// </summary>
public class ListAccounts
{
    /// <summary>
```

```

/// Creates the Organizations client and then calls its
/// ListAccountsAsync method.
/// </summary>
public static async Task Main()
{
    // Create the client object using the default account.
    IAmazonOrganizations client = new AmazonOrganizationsClient();

    var request = new ListAccountsRequest
    {
        MaxResults = 5,
    };

    var response = new ListAccountsResponse();
    try
    {
        do
        {
            response = await client.ListAccountsAsync(request);
            response.Accounts.ForEach(a => DisplayAccounts(a));
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        }
        while (response.NextToken is not null);
    }
    catch (AWSOrganizationsNotInUseException ex)
    {
        Console.WriteLine(ex.Message);
    }
}

/// <summary>
/// Displays information about an Organizations account.
/// </summary>
/// <param name="account">An Organizations account for which to display
/// information on the console.</param>
private static void DisplayAccounts(Account account)
{
    string accountInfo = $"{account.Id} {account.Name}\t{account.Status}";

    Console.WriteLine(accountInfo);
}

```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [ListAccounts](#) à la section Référence des AWS SDK pour .NET API.

ListOrganizationalUnitsForParent

L'exemple de code suivant montre comment utiliser `ListOrganizationalUnitsForParent`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Lists the AWS Organizations organizational units that belong to an
/// organization.
/// </summary>
public class ListOrganizationalUnitsForParent
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// call the ListOrganizationalUnitsForParentAsync method to retrieve
    /// the list of organizational units.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var parentId = "r-0000";
```

```

        var request = new ListOrganizationalUnitsForParentRequest
        {
            ParentId = parentId,
            MaxResults = 5,
        };

        var response = new ListOrganizationalUnitsForParentResponse();
        try
        {
            do
            {
                response = await
client.ListOrganizationalUnitsForParentAsync(request);
                response.OrganizationalUnits.ForEach(u =>
DisplayOrganizationalUnit(u));
                if (response.NextToken is not null)
                {
                    request.NextToken = response.NextToken;
                }
            }
            while (response.NextToken is not null);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    /// <summary>
    /// Displays information about an Organizations organizational unit.
    /// </summary>
    /// <param name="unit">The OrganizationalUnit for which to display
    /// information.</param>
    public static void DisplayOrganizationalUnit(OrganizationalUnit unit)
    {
        string accountInfo = $"{unit.Id} {unit.Name}\t{unit.Arn}";

        Console.WriteLine(accountInfo);
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListOrganizationalUnitsForParent](#) à la section Référence des AWS SDK pour .NET API.

ListPolicies

L'exemple de code suivant montre comment utiliser `ListPolicies`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Organizations;
using Amazon.Organizations.Model;

/// <summary>
/// Shows how to list the AWS Organizations policies associated with an
/// organization.
/// </summary>
public class ListPolicies
{
    /// <summary>
    /// Initializes an Organizations client object, and then calls its
    /// ListPoliciesAsync method.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        // The value for the Filter parameter is required and must must be
        // one of the following:
        //     AISERVICES_OPT_OUT_POLICY
        //     BACKUP_POLICY
        //     SERVICE_CONTROL_POLICY
        //     TAG_POLICY
        var request = new ListPoliciesRequest
```

```

        {
            Filter = "SERVICE_CONTROL_POLICY",
            MaxResults = 5,
        };

        var response = new ListPoliciesResponse();
        try
        {
            do
            {
                response = await client.ListPoliciesAsync(request);
                response.Policies.ForEach(p => DisplayPolicies(p));
                if (response.NextToken is not null)
                {
                    request.NextToken = response.NextToken;
                }
            }
            while (response.NextToken is not null);
        }
        catch (AWSOrganizationsNotInUseException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    /// <summary>
    /// Displays information about the Organizations policies associated
    /// with an organization.
    /// </summary>
    /// <param name="policy">An Organizations policy summary to display
    /// information on the console.</param>
    private static void DisplayPolicies(PolicySummary policy)
    {
        string policyInfo = $"{policy.Id} {policy.Name}\t{policy.Description}";

        Console.WriteLine(policyInfo);
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListPolicies](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation de Partner Central SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide de Partner Central.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

CreateOpportunity

L'exemple de code suivant montre comment utiliser `CreateOpportunity`.

SDK pour .NET

Créez une opportunité.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// PDX-License-Identifier: Apache-2.0

using System;
using Newtonsoft.Json;
using Amazon;
using Amazon.Runtime;
using Amazon.PartnerCentralSelling;
using Amazon.PartnerCentralSelling.Model;

namespace AWSExample
{
    class Program
    {
        static readonly string catalogToUse = "AWS";
        static async Task Main(string[] args)
```

```
{
    // Initialize credentials from .aws/credentials file
    var credentials = new
Amazon.Runtime.CredentialManagement.SharedCredentialsFile();
    if (credentials.TryGetProfile("default", out var profile))
    {
        AWSCredentials awsCredentials =
profile.GetAWSCredentials(credentials);

        var client = new AmazonPartnerCentralSellingClient(awsCredentials);

        var request = new CreateOpportunityRequest
        {
            Catalog = catalogToUse,
            Origin = "Partner Referral",
            Customer = new Customer
            {
                Account = new Account
                {
                    Address = new Address
                    {
                        CountryCode = "US",
                        PostalCode = "99502",
                        StateOrRegion = "Alaska"
                    },
                    CompanyName = "TestCompanyName",
                    Duns = "123456789",
                    WebsiteUrl = "www.test.io",
                    Industry = "Automotive"
                },
                Contacts = new List<Contact>
                {
                    new Contact
                    {
                        Email = "test@test.io",
                        FirstName = "John ",
                        LastName = "Doe",
                        Phone = "+14444444444",
                        BusinessTitle = "test title"
                    }
                }
            },
            Lifecycle = new Lifecycle
            {
```



```

        ReviewStatus = "Submitted",
        TargetCloseDate = "2024-12-30"
    },
    Marketing = new Marketing
    {
        Source = "None"
    },
    OpportunityType = "Net New Business",
    PrimaryNeedsFromAws = new List<string> { "Co-Sell -
Architectural Validation" },
    Project = new Project
    {
        Title = "Moin Test UUID",
        CustomerBusinessProblem = "Sandbox is not working as
expected",

        CustomerUseCase = "AI Machine Learning and Analytics",
        DeliveryModels = new List<string> { "SaaS or PaaS" },
        ExpectedCustomerSpend = new List<ExpectedCustomerSpend>
        {
            new ExpectedCustomerSpend
            {
                Amount = "2000.0",
                CurrencyCode = "USD",
                Frequency = "Monthly",
                TargetCompany = "Ibexlabs"
            }
        },
        SalesActivities = new List<string> { "Initialized
discussions with customer" }
    }
};

try
{
    var response = await client.CreateOpportunityAsync(request);
    Console.WriteLine(response.HttpStatusCode);
    string formattedJson = JsonConvert.SerializeObject(response,
Formatting.Indented);
    Console.WriteLine(formattedJson);
}
catch (ValidationException ex)
{
    Console.WriteLine("Validation error: " + ex.Message);
}

```

```

        catch (AmazonPartnerCentralSellingException e)
        {
            Console.WriteLine("Failed:");
            Console.WriteLine(e.RequestId);
            Console.WriteLine(e.ErrorCode);
            Console.WriteLine(e.Message);
        }
    }
    else
    {
        Console.WriteLine("Profile not found.");
    }
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateOpportunity](#) à la section Référence des AWS SDK pour .NET API.

GetOpportunity

L'exemple de code suivant montre comment utiliser `GetOpportunity`.

SDK pour .NET

Profitez d'une opportunité.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// PDX-License-Identifier: Apache-2.0

using System;
using Newtonsoft.Json;
using Amazon;
using Amazon.Runtime;
using Amazon.PartnerCentralSelling;
using Amazon.PartnerCentralSelling.Model;

namespace AWSExample
{
    class Program
    {
        static readonly string catalogToUse = "AWS";
        static readonly string identifier = "01111111";
    }
}

```

```

static async Task Main(string[] args)
{
    // Initialize credentials from .aws/credentials file
    var credentials = new
Amazon.Runtime.CredentialManagement.SharedCredentialsFile();
    if (credentials.TryGetProfile("default", out var profile))
    {
        AWSCredentials awsCredentials =
profile.GetAWSCredentials(credentials);

        var client = new AmazonPartnerCentralSellingClient(awsCredentials);

        var request = new GetOpportunityRequest
        {
            Catalog = catalogToUse,
            Identifier = identifier
        };

        try {
            var response = await client.GetOpportunityAsync(request);
            Console.WriteLine(response.HttpStatusCode);
            string formattedJson = JsonConvert.SerializeObject(response,
Formatting.Indented);
            Console.WriteLine(formattedJson);
        } catch (ValidationException ex) {
            Console.WriteLine("Validation error: " + ex.Message);
        } catch (AmazonPartnerCentralSellingException e) {
            Console.WriteLine("Failed:");
            Console.WriteLine(e.RequestId);
            Console.WriteLine(e.ErrorCode);
            Console.WriteLine(e.Message);
        }
    }
    else
    {
        Console.WriteLine("Profile not found.");
    }
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [GetOpportunity](#) à la section Référence des AWS SDK pour .NET API.

ListOpportunities

L'exemple de code suivant montre comment utiliser `ListOpportunities`.

SDK pour .NET

Répertoriez les opportunités.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// PDX-License-Identifier: Apache-2.0

using System;
using Newtonsoft.Json;
using Amazon;
using Amazon.Runtime;
using Amazon.PartnerCentralSelling;
using Amazon.PartnerCentralSelling.Model;

namespace AWSExample
{
    class Program
    {
        static readonly string catalogToUse = "Sandbox";
        static async Task Main(string[] args)
        {
            // Initialize credentials from .aws/credentials file
            var credentials = new
Amazon.Runtime.CredentialManagement.SharedCredentialsFile();
            if (credentials.TryGetProfile("default", out var profile))
            {
                AWSCredentials awsCredentials =
profile.GetAWSCredentials(credentials);

                //var config = new AmazonPartnerCentralSellingConfig()
                //{
                //    ServiceURL = "https://partnercentral-selling.us-
east-1.api.aws",
                //};
                //var client = new AmazonPartnerCentralSellingClient(awsCredentials,
config);
```

```

        var client = new AmazonPartnerCentralSellingClient(awsCredentials);
        var request = new ListOpportunitiesRequest
        {
            Catalog = catalogToUse,
            MaxResults = 2
        };

        try {
            var response = await client.ListOpportunitiesAsync(request);
            Console.WriteLine(response.HttpStatusCode);
            foreach (var opportunity in response.OpportunitySummaries)
            {
                Console.WriteLine("Opportunity id: " + opportunity.Id);
            }
            string formattedJson =
                JsonConvert.SerializeObject(response.OpportunitySummaries, Formatting.Indented);
            Console.WriteLine(formattedJson);
        } catch (ValidationException ex) {
            Console.WriteLine("Validation error: " + ex.Message);
        } catch (AmazonPartnerCentralSellingException e) {
            Console.WriteLine("Failed:");
            Console.WriteLine(e.RequestId);
            Console.WriteLine(e.ErrorCode);
            Console.WriteLine(e.Message);
        }
    }
    else
    {
        Console.WriteLine("Profile not found.");
    }
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ListOpportunities](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation d'Amazon Pinpoint SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Pinpoint.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

SendMessage

L'exemple de code suivant montre comment utiliser `SendMessage`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Envoyer un e-mail.

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendEmailMessage;

public class SendEmailMainClass
{
    public static async Task Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
```

```

        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
string region = "us-east-1";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region you're using to send email.
string senderAddress = configuration["SenderAddress"]!;

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
string toAddress = configuration["ToAddress"]!;

// The Amazon Pinpoint project/application ID to use when you send this
message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
string appId = configuration["AppId"]!;

try
{
    await SendEmailMessage(region, appId, toAddress, senderAddress);
}
catch (Exception ex)
{
    Console.WriteLine("The message wasn't sent. Error message: " +
ex.Message);
}

public static async Task<MessageResponse> SendEmailMessage(
    string region, string appId, string toAddress, string senderAddress)
{
    var client = new
AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

// The subject line of the email.
string subject = "Amazon Pinpoint Email test";

// The body of the email for recipients whose email clients don't

```

```

        // support HTML content.
        string textBody = @"Amazon Pinpoint Email Test (.NET)"
            + "\n-----"
            + "\nThis email was sent using the Amazon Pinpoint API
using the AWS SDK for .NET.";

        // The body of the email for recipients whose email clients support
        // HTML content.
        string htmlBody = @"<html>"
            + "\n<head></head>"
            + "\n<body>"
            + "\n  <h1>Amazon Pinpoint Email Test (AWS SDK for .NET)</
h1>"
            + "\n  <p>This email was sent using the "
            + "\n    <a href='https://aws.amazon.com/pinpoint/'>Amazon
Pinpoint</a> API "
            + "\n    using the <a href='https://aws.amazon.com/sdk-
for-net/'>AWS SDK for .NET</a>"
            + "\n  </p>"
            + "\n</body>"
            + "\n</html>";

        // The character encoding the you want to use for the subject line and
        // message body of the email.
        string charset = "UTF-8";

        var sendRequest = new SendMessagesRequest
        {
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses = new Dictionary<string, AddressConfiguration>
                {
                    {
                        toAddress,
                        new AddressConfiguration
                        {
                            ChannelType = ChannelType.EMAIL
                        }
                    }
                },
                MessageConfiguration = new DirectMessageConfiguration
                {
                    EmailMessage = new EmailMessage

```



```

        {
            FromAddress = senderAddress,
            SimpleEmail = new SimpleEmail
            {
                HtmlPart = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = htmlBody
                },
                TextPart = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = textBody
                },
                Subject = new SimpleEmailPart
                {
                    Charset = charset,
                    Data = subject
                }
            }
        }
    }
};
Console.WriteLine("Sending message...");
SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
Console.WriteLine("Message sent!");
return response.MessageResponse;
}
}

```

Envoyer un SMS.

```

using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendSmsMessage;

```

```
public class SendSmsMessageMainClass
{
    public static async Task Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // The AWS Region that you want to use to send the message. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The phone number or short code to send the message from. The phone number
        // or short code that you specify has to be associated with your Amazon
        Pinpoint
        // account. For best results, specify long codes in E.164 format.
        string originationNumber = configuration["OriginationNumber"]!;

        // The recipient's phone number. For best results, you should specify the
        // phone number in E.164 format.
        string destinationNumber = configuration["DestinationNumber"]!;

        // The Pinpoint project/ application ID to use when you send this message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        // The type of SMS message that you want to send. If you plan to send
        // time-sensitive content, specify TRANSACTIONAL. If you plan to send
        // marketing-related content, specify PROMOTIONAL.
        MessageType messageType = MessageType.TRANSACTIONAL;

        // The registered keyword associated with the originating short code.
        string? registeredKeyword = configuration["RegisteredKeyword"];

        // The sender ID to use when sending the message. Support for sender ID
        // varies by country or region. For more information, see
        // https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-
        countries.html
        string? senderId = configuration["SenderId"];
```

```

        try
        {
            var response = await SendSmsMessage(region, appId, destinationNumber,
                originationNumber, registeredKeyword, senderId, messageType);
            Console.WriteLine($"Message sent to
{response.MessageResponse.Result.Count} recipient(s).");
            foreach (var messageResultValue in
                response.MessageResponse.Result.Select(r => r.Value))
            {
                Console.WriteLine($"{messageResultValue.MessageId} Status:
{messageResultValue.DeliveryStatus}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("The message wasn't sent. Error message: " +
ex.Message);
        }
    }

    public static async Task<SendMessagesResponse> SendSmsMessage(
        string region, string appId, string destinationNumber, string
originationNumber,
        string? keyword, string? senderId, MessageType messageType)
    {
        // The content of the SMS message.
        string message = "This message was sent through Amazon Pinpoint using" +
            " the AWS SDK for .NET. Reply STOP to opt out.";

        var client = new
AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

        SendMessagesRequest sendRequest = new SendMessagesRequest
        {
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses =
                    new Dictionary<string, AddressConfiguration>
                    {
                        {

```

```

        destinationNumber,
        new AddressConfiguration { ChannelType =
ChannelType.SMS }
    },
    MessageConfiguration = new DirectMessageConfiguration
    {
        SMSMessage = new SMSMessage
        {
            Body = message,
            MessageType = MessageType.TRANSACTIONAL,
            OriginationNumber = originationNumber,
            SenderId = senderId,
            Keyword = keyword
        }
    }
};
SendMessageResponse response = await client.SendMessageAsync(sendRequest);
return response;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [SendMessage](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation d'Amazon Polly SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Polly.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

DeleteLexicon

L'exemple de code suivant montre comment utiliser `DeleteLexicon`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Deletes an existing Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class DeleteLexicon
{
    public static async Task Main()
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();

        var success = await DeletePollyLexiconAsync(client, lexiconName);

        if (success)
```

```

        {
            Console.WriteLine($"Successfully deleted {lexiconName}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {lexiconName}.");
        }
    }

    /// <summary>
    /// Deletes the named Amazon Polly lexicon.
    /// </summary>
    /// <param name="client">The initialized Amazon Polly client object.</param>
    /// <param name="lexiconName">The name of the Amazon Polly lexicon to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> DeletePollyLexiconAsync(
        AmazonPollyClient client,
        string lexiconName)
    {
        var deleteLexiconRequest = new DeleteLexiconRequest()
        {
            Name = lexiconName,
        };

        var response = await client.DeleteLexiconAsync(deleteLexiconRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteLexicon](#) à la section Référence des AWS SDK pour .NET API.

DescribeVoices

L'exemple de code suivant montre comment utiliser `DescribeVoices`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class DescribeVoices
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();

        var allVoicesRequest = new DescribeVoicesRequest();
        var enUsVoicesRequest = new DescribeVoicesRequest()
        {
            LanguageCode = "en-US",
        };

        try
        {
            string nextToken;
            do
            {
                var allVoicesResponse = await
client.DescribeVoicesAsync(allVoicesRequest);
                nextToken = allVoicesResponse.NextToken;
                allVoicesRequest.NextToken = nextToken;

                Console.WriteLine("\nAll voices: ");
                allVoicesResponse.Voices.ForEach(voice =>
                {
                    DisplayVoiceInfo(voice);
                });
            }
            while (nextToken is not null);
        }
    }
}
```

```

        do
        {
            var enUsVoicesResponse = await
client.DescribeVoicesAsync(enUsVoicesRequest);
            nextToken = enUsVoicesResponse.NextToken;
            enUsVoicesRequest.NextToken = nextToken;

            Console.WriteLine("\nen-US voices: ");
            enUsVoicesResponse.Voices.ForEach(voice =>
            {
                DisplayVoiceInfo(voice);
            });
        }
        while (nextToken is not null);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception caught: " + ex.Message);
    }
}

public static void DisplayVoiceInfo(Voice voice)
{
    Console.WriteLine($" Name: {voice.Name}\tGender:
{voice.Gender}\tLanguageName: {voice.LanguageName}");
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeVoices](#) à la section Référence des AWS SDK pour .NET API.

GetLexicon

L'exemple de code suivant montre comment utiliser `GetLexicon`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Retrieves information about a specific Amazon Polly lexicon.
/// </summary>
public class GetLexicon
{
    public static async Task Main(string[] args)
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();

        await GetPollyLexiconAsync(client, lexiconName);
    }

    public static async Task GetPollyLexiconAsync(AmazonPollyClient client,
string lexiconName)
    {
        var getLexiconRequest = new GetLexiconRequest()
        {
            Name = lexiconName,
        };

        try
        {
            var response = await client.GetLexiconAsync(getLexiconRequest);
            Console.WriteLine($"Lexicon:\n Name: {response.Lexicon.Name}");
            Console.WriteLine($"Content: {response.Lexicon.Content}");
        }
        catch (Exception ex)
```

```
        {  
            Console.WriteLine("Error: " + ex.Message);  
        }  
    }  
}
```

- Pour plus de détails sur l'API, reportez-vous [GetLexicon](#) à la section Référence des AWS SDK pour .NET API.

ListLexicons

L'exemple de code suivant montre comment utiliser `ListLexicons`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Polly;  
using Amazon.Polly.Model;  
  
/// <summary>  
/// Lists the Amazon Polly lexicons that have been defined. By default,  
/// lists the lexicons that are defined in the same AWS Region as the default  
/// user. To view Amazon Polly lexicons that are defined in a different AWS  
/// Region, supply it as a parameter to the Amazon Polly constructor.  
/// </summary>  
public class ListLexicons  
{  
    public static async Task Main()  
    {  
        var client = new AmazonPollyClient();  
        var request = new ListLexiconsRequest();
```

```

    try
    {
        Console.WriteLine("All voices: ");

        do
        {
            var response = await client.ListLexiconsAsync(request);
            request.NextToken = response.NextToken;

            response.Lexicons.ForEach(lexicon =>
            {
                var attributes = lexicon.Attributes;
                Console.WriteLine($"Name: {lexicon.Name}");
                Console.WriteLine($"\\tAlphabet: {attributes.Alphabet}");
                Console.WriteLine($"\\tLanguageCode:
{attributes.LanguageCode}");
                Console.WriteLine($"\\tLastModified:
{attributes.LastModified}");
                Console.WriteLine($"\\tLexemesCount:
{attributes.LexemesCount}");
                Console.WriteLine($"\\tLexiconArn: {attributes.LexiconArn}");
                Console.WriteLine($"\\tSize: {attributes.Size}");
            });
        }
        while (request.NextToken is not null);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListLexicons](#) à la section Référence des AWS SDK pour .NET API.

PutLexicon

L'exemple de code suivant montre comment utiliser `PutLexicon`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Creates a new Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class PutLexicon
{
    public static async Task Main()
    {
        string lexiconContent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
            "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/" +
            "pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
            "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-" +
            "lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
            "alphabet=\"ipa\" xml:lang=\"en-US\">" +
            "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>" +
            "</lexicon>";
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();
        var putLexiconRequest = new PutLexiconRequest()
        {
            Name = lexiconName,
            Content = lexiconContent,
        };

        try
        {
            var response = await client.PutLexiconAsync(putLexiconRequest);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {

```

```
        Console.WriteLine($"Successfully created Lexicon:
{lexiconName}.");
    }
    else
    {
        Console.WriteLine($"Could not create Lexicon: {lexiconName}.");
    }
}
catch (Exception ex)
{
    Console.WriteLine("Exception caught: " + ex.Message);
}
}
```

- Pour plus de détails sur l'API, reportez-vous [PutLexicon](#) à la section Référence des AWS SDK pour .NET API.

SynthesizeSpeech

L'exemple de code suivant montre comment utiliser SynthesizeSpeech.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeech
{
    public static async Task Main()
```

```

    {
        string outputFileName = "speech.mp3";
        string text = "Twas brillig, and the slithy toves did gyre and gimbol in
the wabe";

        var client = new AmazonPollyClient();
        var response = await PollySynthesizeSpeech(client, text);

        WriteSpeechToStream(response.AudioStream, outputFileName);
    }

    /// <summary>
    /// Calls the Amazon Polly SynthesizeSpeechAsync method to convert text
    /// to speech.
    /// </summary>
    /// <param name="client">The Amazon Polly client object used to connect
    /// to the Amazon Polly service.</param>
    /// <param name="text">The text to convert to speech.</param>
    /// <returns>A SynthesizeSpeechResponse object that includes an AudioStream
    /// object with the converted text.</returns>
    private static async Task<SynthesizeSpeechResponse>
PollySynthesizeSpeech(IAmazonPolly client, string text)
    {
        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
            OutputFormat = OutputFormat.Mp3,
            VoiceId = VoiceId.Joanna,
            Text = text,
        };

        var synthesizeSpeechResponse =
            await client.SynthesizeSpeechAsync(synthesizeSpeechRequest);

        return synthesizeSpeechResponse;
    }

    /// <summary>
    /// Writes the AudioStream returned from the call to
    /// SynthesizeSpeechAsync to a file in MP3 format.
    /// </summary>
    /// <param name="audioStream">The AudioStream returned from the
    /// call to the SynthesizeSpeechAsync method.</param>
    /// <param name="outputFileName">The full path to the file in which to
    /// save the audio stream.</param>

```

```
private static void WriteSpeechToStream(Stream audioStream, string
outputFileName)
{
    var outputStream = new FileStream(
        outputFileName,
        FileMode.Create,
        FileAccess.Write);
    byte[] buffer = new byte[2 * 1024];
    int readBytes;

    while ((readBytes = audioStream.Read(buffer, 0, 2 * 1024)) > 0)
    {
        outputStream.Write(buffer, 0, readBytes);
    }

    // Flushes the buffer to avoid losing the last second or so of
    // the synthesized text.
    outputStream.Flush();
    Console.WriteLine($"Saved {outputFileName} to disk.");
}
}
```

Synthétisez la parole à partir du texte à l'aide de marques vocales avec Amazon Polly à l'aide d'AWS un SDK.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeechMarks
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        string outputFileName = "speechMarks.json";

        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
```

```

        OutputFormat = OutputFormat.Json,
        SpeechMarkTypes = new List<string>
        {
            SpeechMarkType.Viseme,
            SpeechMarkType.Word,
        },
        VoiceId = VoiceId.Joanna,
        Text = "This is a sample text to be synthesized.",
    };

    try
    {
        using (var outputStream = new FileStream(outputFileName,
            FileMode.Create, FileAccess.Write))
        {
            var synthesizeSpeechResponse = await
client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
            var buffer = new byte[2 * 1024];
            int readBytes;

            var inputStream = synthesizeSpeechResponse.AudioStream;
            while ((readBytes = inputStream.Read(buffer, 0, 2 * 1024)) > 0)
            {
                outputStream.Write(buffer, 0, readBytes);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [SynthesizeSpeech](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemples d'Amazon RDS utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon RDS.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)

Mise en route

Hello Amazon RDS

L'exemple de code suivant montre comment commencer à utiliser Amazon RDS.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.RDS;
using Amazon.RDS.Model;
```

```
namespace RDSActions;

public static class HelloRds
{
    static async Task Main(string[] args)
    {
        var rdsClient = new AmazonRDSClient();

        Console.WriteLine($"Hello Amazon RDS! Following are some of your DB instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first twenty DB instances.
        var response = await rdsClient.DescribeDBInstancesAsync(
            new DescribeDBInstancesRequest()
            {
                MaxRecords = 20 // Must be between 20 and 100.
            });

        foreach (var instance in response.DBInstances)
        {
            Console.WriteLine($"  \tDB name: {instance.DBName}");
            Console.WriteLine($"  \tArn: {instance.DBInstanceArn}");
            Console.WriteLine($"  \tIdentifier: {instance.DBInstanceIdentifier}");
            Console.WriteLine();
        }
    }
}
```

- Pour plus de détails sur l'API, voir [Description DBInstances](#) dans le manuel de référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Créez un groupe de paramètres de bases de données personnalisé et définissez des valeurs pour les paramètres.

- Créez une instance de base de données configurée pour utiliser le groupe de paramètres. L'instance de base de données contient également une base de données.
- Prenez un instantané de l'instance.
- Supprimez l'instance et le groupe de paramètres.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET example performs the following tasks:
    1. Returns a list of the available DB engine families using the
    DescribeDBEngineVersionsAsync method.
    2. Selects an engine family and creates a custom DB parameter group using the
    CreateDBParameterGroupAsync method.
    3. Gets the parameter groups using the DescribeDBParameterGroupsAsync method.
    4. Gets parameters in the group using the DescribeDBParameters method.
    5. Parses and displays parameters in the group.
    6. Modifies both the auto_increment_offset and auto_increment_increment
    parameters
    using the ModifyDBParameterGroupAsync method.
    7. Gets and displays the updated parameters using the DescribeDBParameters
    method with a source of "user".
    8. Gets a list of allowed engine versions using the
    DescribeDBEngineVersionsAsync method.
```

9. Displays and selects from a list of micro instance classes available for the selected engine and version.
 10. Creates an RDS DB instance that contains a MySQL database and uses the parameter group using the `CreateDBInstanceAsync` method.
 11. Waits for DB instance to be ready using the `DescribeDBInstancesAsync` method.
 12. Prints out the connection endpoint string for the new DB instance.
 13. Creates a snapshot of the DB instance using the `CreateDBSnapshotAsync` method.
 14. Waits for DB snapshot to be ready using the `DescribeDBSnapshots` method.
 15. Deletes the DB instance using the `DeleteDBInstanceAsync` method.
 16. Waits for DB instance to be deleted using the `DescribeDbInstances` method.
 17. Deletes the parameter group using the `DeleteDBParameterGroupAsync`.
- */

```
private static readonly string sepBar = new('-', 80);
private static RDSWrapper rdsWrapper = null!;
private static ILogger logger = null!;
private static readonly string engine = "mysql";
static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon RDS service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonRDS>()
                .AddTransient<RDSWrapper>()
        )
        .Build();

    logger = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
    }).CreateLogger<RDSInstanceScenario>();

    rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

    Console.WriteLine(sepBar);
    Console.WriteLine(
```

```
        "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
        Console.WriteLine(sepBar);

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamily();

            var parameterGroup = await CreateDbParameterGroup(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParameters(parameterGroup.DBParameterGroupName, parameters);

            await DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

            var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

            var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
                instanceChoice.DBInstanceClass, newInstanceIdentifier);
            if (newInstance != null)
            {
                DisplayConnectionString(newInstance);

                await CreateSnapshot(newInstance);

                await DeleteRdsInstance(newInstance);
            }

            await DeleteParameterGroup(parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
    }
```

```

        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the RDS DB parameter group family from a list of available options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamily()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

        Console.WriteLine("1. The following is a list of available DB parameter
group families:");
        int i = 1;
        var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
        foreach (var parameterGroupFamily in parameterGroupFamilies)
        {
            // List the available parameter group families.
            Console.WriteLine(
                $"{i}\t{i}. Family: {parameterGroupFamily.Key}");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.

```

```

    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName);

        var matchingParameters =

```



```

        parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"{p.ParameterName}." +
                $"{p.Description}." +
                $"{p.AllowedValues}." +
                $"{p.ParameterValue}"));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                int newValue = 0;
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    Int32.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }
    }

```

```

    }
}

await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

Console.WriteLine(sepBar);
}

/// <summary>
/// Describe the user source parameters in the group.
/// </summary>
/// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
/// <returns>Async task.</returns>
public static async Task DescribeUserSourceParameters(string parameterGroupName)
{
    Console.WriteLine(sepBar);
    Console.WriteLine("7. Describe user source parameters in the group.");

    var parameters =
        await rdsWrapper.DescribeDBParameters(parameterGroupName, "user");

    parameters.ForEach(p =>
        Console.WriteLine(
            $"{p.ParameterName}." +
            $"{p.Description}." +
            $"{p.AllowedValues}." +
            $"{p.ParameterValue}."));

    Console.WriteLine(sepBar);
}

/// <summary>
/// Choose a DB engine version.
/// </summary>
/// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
/// <returns>The selected engine version.</returns>
public static async Task<DBEngineVersion> ChooseDbEngineVersion(string
dbParameterGroupFamily)
{
    Console.WriteLine(sepBar);
    // Get a list of allowed engines.

```

```

        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\\t{i}. Engine: {version.Engine} Version
{version.EngineVersion}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDbInstanceClass(string
engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
engineVersion);

```

```

        Console.WriteLine($"8. Available micro DB instance classes for engine
{engine} and version {engineVersion}:");
        int i = 1;

        // Filter to micro instances for this example.
        allowedInstances = allowedInstances
            .Where(i => i.DBInstanceClass.Contains("micro")).ToList();

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\\t{i}. Instance class: {instance.DBInstanceClass} (storage type
{instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("9. Select an available DB instance class by entering
a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB instance.</
param>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
instance.</param>
    /// <returns>The new DB instance.</returns>

```

```
public static async Task<DBInstance?> CreateRdsNewInstance(DBParameterGroup
parameterGroup,
    string engineName, string engineVersion, string instanceClass, string
instanceIdentifier)
{
    Console.WriteLine(sepBar);
    Console.WriteLine($"10. Create a new DB instance with identifier
{instanceIdentifier}.");
    bool isInstanceReady = false;
    DBInstance newInstance;
    var instances = await rdsWrapper.DescribeDBInstances();
    isInstanceReady = instances.FirstOrDefault(i =>
        i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

    if (isInstanceReady)
    {
        Console.WriteLine("Instance already created.");
        newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
    }
    else
    {
        Console.WriteLine("Please enter an admin user name:");
        var username = Console.ReadLine();

        Console.WriteLine("Please enter an admin password:");
        var password = Console.ReadLine();

        newInstance = await rdsWrapper.CreateDBInstance(
            "ExampleInstance",
            instanceIdentifier,
            parameterGroup.DBParameterGroupName,
            engineName,
            engineVersion,
            instanceClass,
            20,
            username,
            password
        );

        // 11. Wait for the DB instance to be ready.

        Console.WriteLine("11. Waiting for DB instance to be ready...");
    }
}
```

```

        while (!isInstanceReady)
        {
            instances = await
rdsWrapper.DescribeDBInstances(instanceIdentifier);
            isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
            newInstance = instances.First();
            Thread.Sleep(30000);
        }
    }

    Console.WriteLine(sepBar);
    return newInstance;
}

/// <summary>
/// Display a connection string for an RDS DB instance.
/// </summary>
/// <param name="instance">The DB instance to use to get a connection string.</
param>
public static void DisplayConnectionString(DBInstance instance)
{
    Console.WriteLine(sepBar);
    // Display the connection string.
    Console.WriteLine("12. New DB instance connection string: ");
    Console.WriteLine(
        $"{engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
"
        + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");

    Console.WriteLine(sepBar);
}

/// <summary>
/// Create a snapshot from an RDS DB instance.
/// </summary>
/// <param name="instance">DB instance to use when creating a snapshot.</param>
/// <returns>The snapshot object.</returns>
public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
{
    Console.WriteLine(sepBar);
    // Create a snapshot.
    Console.WriteLine($"13. Creating snapshot from DB instance
{instance.DBInstanceIdentifier}.");
}

```

```

        var snapshot = await
rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            var snapshots = await
rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
    {
        Console.WriteLine(sepBar);
        // Delete the DB instance.
        Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
        await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);

        // Wait for the DB instance to delete.
        Console.WriteLine($"16. Waiting for the DB instance to delete...");
        bool isInstanceDeleted = false;

        while (!isInstanceDeleted)
        {
            var instance = await rdsWrapper.DescribeDBInstances();

```

```

        isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
newInstance.DBInstanceIdentifier);
        Thread.Sleep(30000);
    }

    Console.WriteLine("DB instance deleted.");
    Console.WriteLine(sepBar);
}

/// <summary>
/// Delete a DB parameter group.
/// </summary>
/// <param name="parameterGroup">The parameter group to delete.</param>
/// <returns>Async task.</returns>
public static async Task DeleteParameterGroup(DBParameterGroup parameterGroup)
{
    Console.WriteLine(sepBar);
    // Delete the parameter group.
    Console.WriteLine($"17. Delete the DB parameter group
{parameterGroup.DBParameterGroupName}.");
    await
rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

    Console.WriteLine(sepBar);
}

```

Méthodes d'encapsulation utilisées par le scénario pour les actions d'instance de base de données.

```

/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with DB
instance operations.
/// </summary>
public partial class RDSWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public RDSWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }
}

```



```

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
        _amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
            paginateInstanceOptions.OrderableDBInstanceOptions)

```

```

        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstances(string
dbInstanceIdIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdIdentifier = dbInstanceIdIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Create an RDS DB instance with a particular set of properties. Use the
action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbName">Name for the DB instance.</param>
    /// <param name="dbInstanceIdIdentifier">DB instance identifier.</param>
    /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>

```

```

    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
    /// <param name="adminName">Admin user name.</param>
    /// <param name="adminPassword">Admin user password.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceId,
        string parameterGroupName, string dbEngine, string dbEngineVersion,
        string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
    {
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBName = dbName,
                DBInstanceId = dbInstanceId,
                DBParameterGroupName = parameterGroupName,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass,
                AllocatedStorage = allocatedStorage,
                MasterUsername = adminName,
                MasterUserPassword = adminPassword
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceId">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstance(string dbInstanceId)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceId = dbInstanceId,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });
    }

```

```

        });

        return response.DBInstance;
    }

```

Méthodes d'encapsulation utilisées par le scénario pour les groupes de paramètres de base de données.

```

/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
/// parameter groups.
/// </summary>
public partial class RDSWrapper
{
    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
= null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }

    /// <summary>
    /// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>

```

```

    /// <param name="family">Family of the DB parameter group.</param>
    /// <param name="description">Description of the DB parameter group.</param>
    /// <returns>The new DB parameter group.</returns>
    public async Task<DBParameterGroup> CreateDBParameterGroup(
        string name, string family, string description)
    {
        var response = await _amazonRDS.CreateDBParameterGroupAsync(
            new CreateDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                DBParameterGroupFamily = family,
                Description = description
            });
        return response.DBParameterGroup;
    }

    /// <summary>
    /// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }

    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
group
    /// or be associated with any DB instances.

```

```

    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
}

```

Méthodes d'encapsulation utilisées par le scénario pour les actions d'instantané de base de données.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
/// snapshots.
/// </summary>
public partial class RDSWrapper
{
    /// <summary>
    /// Create a snapshot of a DB instance.
    /// </summary>
    /// <param name="dbInstanceIdIdentifier">DB instance identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdIdentifier,
string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdIdentifier = dbInstanceIdIdentifier
            });

        return response.DBSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdIdentifier">DB instance identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdIdentifier)
    {
        var results = new List<DBSnapshot>();
        var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
            new DescribeDBSnapshotsRequest()
```

```
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });

        // Get the entire list using the paginator.
        await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
        {
            results.Add(snapshots);
        }
        return results;
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CréerDBInstance](#)
 - [Créer un DBParameter groupe](#)
 - [CréerDBSnapshot](#)
 - [SuppressionDBInstance](#)
 - [Supprimer le DBParameter groupe](#)
 - [Décrire DBEngine les versions](#)
 - [DécrireDBInstances](#)
 - [Décrire DBParameter les groupes](#)
 - [DécrireDBParameters](#)
 - [DécrireDBSnapshots](#)
 - [DescribeOrderableDBInstanceOptions](#)
 - [Modifier le DBParameter groupe](#)

Actions

CreateDBInstance

L'exemple de code suivant montre comment utiliser CreateDBInstance.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Create an RDS DB instance with a particular set of properties. Use the
    action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbName">Name for the DB instance.</param>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="parameterGroupName">DB parameter group to associate with the
    instance.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
    allocate to the DB instance.</param>
    /// <param name="adminName">Admin user name.</param>
    /// <param name="adminPassword">Admin user password.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstance(string dbName, string
    dbInstanceIdentifier,
        string parameterGroupName, string dbEngine, string dbEngineVersion,
        string instanceClass, int allocatedStorage, string adminName, string
    adminPassword)
    {
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBName = dbName,
                DBInstanceIdentifier = dbInstanceIdentifier,
                DBParameterGroupName = parameterGroupName,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass,
                AllocatedStorage = allocatedStorage,

```

```

        MasterUsername = adminName,
        MasterUserPassword = adminPassword
    });

    return response.DBInstance;
}

```

- Pour plus de détails sur l'API, consultez [Create DBInstance](#) in AWS SDK pour .NET API Reference.

CreateDBParameterGroup

L'exemple de code suivant montre comment utiliser `CreateDBParameterGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
/// to determine when the DB parameter group is ready to use.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <param name="family">Family of the DB parameter group.</param>
/// <param name="description">Description of the DB parameter group.</param>
/// <returns>The new DB parameter group.</returns>
public async Task<DBParameterGroup> CreateDBParameterGroup(
    string name, string family, string description)
{
    var response = await _amazonRDS.CreateDBParameterGroupAsync(
        new CreateDBParameterGroupRequest()
        {
            DBParameterGroupName = name,

```

```

        DBParameterGroupFamily = family,
        Description = description
    });
    return response.DBParameterGroup;
}

```

- Pour plus de détails sur l'API, consultez la section [Créer un DBParameter groupe](#) dans le manuel de référence des AWS SDK pour .NET API.

CreateDBSnapshot

L'exemple de code suivant montre comment utiliser CreateDBSnapshot.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create a snapshot of a DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
/// <returns>DB snapshot object.</returns>
public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
{
    var response = await _amazonRDS.CreateDBSnapshotAsync(
        new CreateDBSnapshotRequest()
        {
            DBSnapshotIdentifier = snapshotIdentifier,
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    return response.DBSnapshot;
}

```

```
}
```

- Pour plus de détails sur l'API, consultez [Create DBSnapshot](#) in AWS SDK pour .NET API Reference.

DeleteDBInstance

L'exemple de code suivant montre comment utiliser `DeleteDBInstance`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}
```

- Pour plus de détails sur l'API, voir [Supprimer DBInstance dans le](#) manuel de référence des AWS SDK pour .NET API.

DeleteDBParameterGroup

L'exemple de code suivant montre comment utiliser `DeleteDBParameterGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a DB parameter group. The group cannot be a default DB parameter
group
/// or be associated with any DB instances.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteDBParameterGroup(string name)
{
    var response = await _amazonRDS.DeleteDBParameterGroupAsync(
        new DeleteDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, voir [Supprimer le DBParameter groupe](#) dans le manuel de référence des AWS SDK pour .NET API.

DescribeDBEngineVersions

L'exemple de code suivant montre comment utiliser `DescribeDBEngineVersions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of DB engine versions for a particular DB engine.
/// </summary>
/// <param name="engine">Name of the engine.</param>
/// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
/// <returns>List of DBEngineVersions.</returns>
public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
    string dbParameterGroupFamily = null)
{
    var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
        new DescribeDBEngineVersionsRequest()
        {
            Engine = engine,
            DBParameterGroupFamily = dbParameterGroupFamily
        });
    return response.DBEngineVersions;
}
```

- Pour plus de détails sur l'API, consultez la section [Décrire DBEngine les versions](#) dans le AWS SDK pour .NET manuel de référence des API.

DescribeDBInstances

L'exemple de code suivant montre comment utiliser `DescribeDBInstances`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Returns a list of DB instances.
/// </summary>
/// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
/// <returns>List of DB instances.</returns>
public async Task<List<DBInstance>> DescribeDBInstances(string
dbInstanceIdentifier = null)
{
    var results = new List<DBInstance>();
    var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
        new DescribeDBInstancesRequest
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });
    // Get the entire list using the paginator.
    await foreach (var instances in instancesPaginator.DBInstances)
    {
        results.Add(instances);
    }
    return results;
}
```

- Pour plus de détails sur l'API, voir [Description DBInstances](#) dans le manuel de référence des AWS SDK pour .NET API.

DescribeDBParameterGroups

L'exemple de code suivant montre comment utiliser `DescribeDBParameterGroups`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get descriptions of DB parameter groups.
/// </summary>
/// <param name="name">Optional name of the DB parameter group to describe.</
param>
/// <returns>The list of DB parameter group descriptions.</returns>
public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
= null)
{
    var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
        new DescribeDBParameterGroupsRequest()
        {
            DBParameterGroupName = name
        });
    return response.DBParameterGroups;
}
```

- Pour plus de détails sur l'API, consultez la section [Décrire DBParameter les groupes](#) dans le AWS SDK pour .NET manuel de référence des API.

DescribeDBParameters

L'exemple de code suivant montre comment utiliser `DescribeDBParameters`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of DB parameters from a specific parameter group.
/// </summary>
/// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
/// <param name="source">Optional source for selecting parameters.</param>
/// <returns>List of parameter values.</returns>
public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
{
    var results = new List<Parameter>();
    var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
        new DescribeDBParametersRequest()
        {
            DBParameterGroupName = dbParameterGroupName,
            Source = source
        });
    // Get the entire list using the paginator.
    await foreach (var parameters in paginateParameters.Parameters)
    {
        results.Add(parameters);
    }
    return results;
}
```

- Pour plus de détails sur l'API, voir [Description DBParameters](#) dans le manuel de référence des AWS SDK pour .NET API.

DescribeDBSnapshots

L'exemple de code suivant montre comment utiliser `DescribeDBSnapshots`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Return a list of DB snapshots for a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>List of DB snapshots.</returns>
public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
{
    var results = new List<DBSnapshot>();
    var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
        new DescribeDBSnapshotsRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    // Get the entire list using the paginator.
    await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
    {
        results.Add(snapshots);
    }
    return results;
}
```

- Pour plus de détails sur l'API, voir [Description DBSnapshots](#) dans le manuel de référence des AWS SDK pour .NET API.

DescribeOrderableDBInstanceOptions

L'exemple de code suivant montre comment utiliser `DescribeOrderableDBInstanceOptions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of orderable DB instance options for a specific
/// engine and engine version.
/// </summary>
/// <param name="engine">Name of the engine.</param>
/// <param name="engineVersion">Version of the engine.</param>
/// <returns>List of OrderableDBInstanceOptions.</returns>
public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
{
    // Use a paginator to get a list of DB instance options.
    var results = new List<OrderableDBInstanceOption>();
    var paginateInstanceOptions =
        _amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
    // Get the entire list using the paginator.
    await foreach (var instanceOptions in
        paginateInstanceOptions.OrderableDBInstanceOptions)
    {
        results.Add(instanceOptions);
    }
    return results;
}
```

- Pour plus de détails sur l'API, consultez la section [DescribeOrderableDBInstanceOptions](#) du manuel de référence des AWS SDK pour .NET API.

ModifyDBParameterGroup

L'exemple de code suivant montre comment utiliser `ModifyDBParameterGroup`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
/// to determine when the DB parameter group is ready to use.
/// </summary>
/// <param name="name">Name of the DB parameter group.</param>
/// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
/// <returns>The updated DB parameter group name.</returns>
public async Task<string> ModifyDBParameterGroup(
    string name, List<Parameter> parameters)
{
    var response = await _amazonRDS.ModifyDBParameterGroupAsync(
        new ModifyDBParameterGroupRequest()
        {
            DBParameterGroupName = name,
            Parameters = parameters,
        });
    return response.DBParameterGroupName;
}
```

- Pour plus de détails sur l'API, voir [Modifier DBParameter le groupe](#) dans la référence des AWS SDK pour .NET API.

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser le AWS SDK pour .NET pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front-end créé avec React.js pour interagir avec un backend RESTful .NET.

- Intégrez une application Web React à AWS des services.
- Listez, ajoutez et mettez à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés à l'aide d'Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples sans serveur

Connexion à une base de données Amazon RDS dans une fonction Lambda

L'exemple de code suivant montre comment implémenter une fonction Lambda qui se connecte à une base de données RDS. La fonction effectue une simple requête de base de données et renvoie le résultat.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Connexion à une base de données Amazon RDS dans une fonction Lambda à l'aide de .NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
    /// Handles the Lambda function execution for connecting to RDS using IAM
    authentication.
    /// </summary>
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
    FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
```

```

{
    // Sample Input: {"body": "{\"key1\": \"20\", \"key2\": \"25\"}"}
    var input = JsonSerializer.Deserialize<InputModel>(request.Body);

    /// Obtain authentication token
    var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
        Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
        Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
        Environment.GetEnvironmentVariable("RDS_USERNAME")
    );

    /// Build the Connection String with the Token
    string connectionString =
        $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +
        $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +
        $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
        $"Pwd={authToken}";

    try
    {
        await using var connection = new MySqlConnection(connectionString);
        await connection.OpenAsync();

        const string sql = "SELECT @param1 + @param2 AS Sum";

        await using var command = new MySqlCommand(sql, connection);
        command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
"0"));
        command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
"0"));

        await using var reader = await command.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            int result = reader.GetInt32("Sum");

            //Sample Response: {"statusCode":200,"body":{"\"message\": \"The sum
is: 45\"},"isBase64Encoded":false}
            return new APIGatewayProxyResponse
            {
                StatusCode = 200,

```

```
        Body = JsonSerializer.Serialize(new { message = $"The sum is: {result}" });
    };
}

}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}

return new APIGatewayProxyResponse
{
    StatusCode = 500,
    Body = JsonSerializer.Serialize(new { error = "Internal server error" })
};
}
}
```

Exemples d'Amazon RDS Data Service utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon RDS Data Service.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Scénarios](#)

Scénarios

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser le AWS SDK pour .NET pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front-end créé avec React.js pour interagir avec un backend RESTful .NET.

- Intégrez une application Web React à AWS des services.
- Listez, ajoutez et mettez à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés à l'aide d'Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Exemples d'utilisation d'Amazon Rekognition SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Rekognition.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

CompareFaces

L'exemple de code suivant montre comment utiliser CompareFaces.

Pour plus d'informations, consultez [Comparaison de visages dans des images](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to compare faces in two images.
/// </summary>
public class CompareFaces
{
    public static async Task Main()
    {
```

```
float similarityThreshold = 70F;
string sourceImage = "source.jpg";
string targetImage = "target.jpg";

var rekognitionClient = new AmazonRekognitionClient();

Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

    try
    {
        using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
        byte[] data = new byte[fs.Length];
        fs.Read(data, 0, (int)fs.Length);
        imageSource.Bytes = new MemoryStream(data);
    }
    catch (Exception)
    {
        Console.WriteLine($"Failed to load source image: {sourceImage}");
        return;
    }

    Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

    try
    {
        using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
        byte[] data = new byte[fs.Length];
        data = new byte[fs.Length];
        fs.Read(data, 0, (int)fs.Length);
        imageTarget.Bytes = new MemoryStream(data);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Failed to load target image: {targetImage}");
        Console.WriteLine(ex.Message);
        return;
    }

var compareFacesRequest = new CompareFacesRequest
{
```

```
        SourceImage = imageSource,
        TargetImage = imageTarget,
        SimilarityThreshold = similarityThreshold,
    };

    // Call operation
    var compareFacesResponse = await
    rekognitionClient.CompareFacesAsync(compareFacesRequest);

    // Display results
    compareFacesResponse.FaceMatches.ForEach(match =>
    {
        ComparedFace face = match.Face;
        BoundingBox position = face.BoundingBox;
        Console.WriteLine($"Face at {position.Left} {position.Top} matches
with {match.Similarity}% confidence.");
    });

    Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
face(s) that did not match.");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CompareFaces](#) à la section Référence des AWS SDK pour .NET API.

CreateCollection

L'exemple de code suivant montre comment utiliser `CreateCollection`.

Pour plus d'informations, consultez [Création d'une collection](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses Amazon Rekognition to create a collection to which you can add
/// faces using the IndexFaces operation.
/// </summary>
public class CreateCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Creating collection: " + collectionId);

        var createCollectionRequest = new CreateCollectionRequest
        {
            CollectionId = collectionId,
        };

        CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
        Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
        Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateCollection](#) à la section Référence des AWS SDK pour .NET API.

DeleteCollection

L'exemple de code suivant montre comment utiliser `DeleteCollection`.

Pour plus d'informations, consultez [Suppression d'une collection](#).

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to delete an existing collection.
/// </summary>
public class DeleteCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Deleting collection: " + collectionId);

        var deleteCollectionRequest = new DeleteCollectionRequest()
        {
            CollectionId = collectionId,
        };

        var deleteCollectionResponse = await
            rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
        Console.WriteLine($"{collectionId}:
{deleteCollectionResponse.StatusCode}");
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteCollection](#) à la section Référence des AWS SDK pour .NET API.

DeleteFaces

L'exemple de code suivant montre comment utiliser `DeleteFaces`.

Pour plus d'informations, veuillez consulter [Supprimer des visages d'une collection](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to delete one or more faces from
/// a Rekognition collection.
/// </summary>
public class DeleteFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx" };

        var rekognitionClient = new AmazonRekognitionClient();

        var deleteFacesRequest = new DeleteFacesRequest()
        {
            CollectionId = collectionId,
            FaceIds = faces,
        };

        DeleteFacesResponse deleteFacesResponse = await
        rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
        deleteFacesResponse.DeletedFaces.ForEach(face =>
        {
```

```
        Console.WriteLine($"FaceID: {face}");  
    });  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteFaces](#) à la section Référence des AWS SDK pour .NET API.

DescribeCollection

L'exemple de code suivant montre comment utiliser `DescribeCollection`.

Pour plus d'informations, veuillez consulter [Description d'une collection](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Rekognition;  
using Amazon.Rekognition.Model;  
  
/// <summary>  
/// Uses the Amazon Rekognition Service to describe the contents of a  
/// collection.  
/// </summary>  
public class DescribeCollection  
{  
    public static async Task Main()  
    {  
        var rekognitionClient = new AmazonRekognitionClient();  
  
        string collectionId = "MyCollection";  
        Console.WriteLine($"Describing collection: {collectionId}");  
    }  
}
```



```
var describeCollectionRequest = new DescribeCollectionRequest()
{
    CollectionId = collectionId,
};

var describeCollectionResponse = await
rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
Console.WriteLine($"Collection ARN:
{describeCollectionResponse.CollectionARN}");
Console.WriteLine($"Face count:
{describeCollectionResponse.FaceCount}");
Console.WriteLine($"Face model version:
{describeCollectionResponse.FaceModelVersion}");
Console.WriteLine($"Created:
{describeCollectionResponse.CreationTimestamp}");
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeCollection](#) à la section Référence des AWS SDK pour .NET API.

DetectFaces

L'exemple de code suivant montre comment utiliser `DetectFaces`.

Pour plus d'informations, veuillez consulter [Détecter des visages dans une image](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectFaces
{
    public static async Task Main()
    {
        string photo = "input.jpg";
        string bucket = "amzn-s3-demo-bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectFacesRequest = new DetectFacesRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },

            // Attributes can be "ALL" or "DEFAULT".
            // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
            // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
            // items/Rekognition/TFaceDetail.html
            Attributes = new List<string>() { "ALL" },
        };

        try
        {
            DetectFacesResponse detectFacesResponse = await
            rekognitionClient.DetectFacesAsync(detectFacesRequest);
            bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
            foreach (FaceDetail face in detectFacesResponse.FaceDetails)
            {
                Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
                left={face.BoundingBox.Top} width={face.BoundingBox.Width}
                height={face.BoundingBox.Height}");
                Console.WriteLine($"Confidence: {face.Confidence}");
                Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
            }
        }
    }
}

```

```

        Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
        Console.WriteLine($"Brightness:
{face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

        if (hasAll)
        {
            Console.WriteLine($"Estimated age is between
{face.AgeRange.Low} and {face.AgeRange.High} years old.");
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
}

```

Afficher les informations du cadre de délimitation pour tous les visages d'une image.

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to display the details of the
/// bounding boxes around the faces detected in an image.
/// </summary>
public class ImageOrientationBoundingBox
{
    public static async Task Main()
    {
        string photo = @"D:\Development\AWS-Examples\Rekognition\target.jpg"; //
"photo.jpg";

        var rekognitionClient = new AmazonRekognitionClient();
    }
}

```

```
var image = new Amazon.Rekognition.Model.Image();
try
{
    using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
    byte[] data = null;
    data = new byte[fs.Length];
    fs.Read(data, 0, (int)fs.Length);
    image.Bytes = new MemoryStream(data);
}
catch (Exception)
{
    Console.WriteLine("Failed to load file " + photo);
    return;
}

int height;
int width;

// Used to extract original photo width/height
using (var imageBitmap = new Bitmap(photo))
{
    height = imageBitmap.Height;
    width = imageBitmap.Width;
}

Console.WriteLine("Image Information:");
Console.WriteLine(photo);
Console.WriteLine("Image Height: " + height);
Console.WriteLine("Image Width: " + width);

try
{
    var detectFacesRequest = new DetectFacesRequest()
    {
        Image = image,
        Attributes = new List<string>() { "ALL" },
    };

    DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
    detectFacesResponse.FaceDetails.ForEach(face =>
    {
```

```

        Console.WriteLine("Face:");
        ShowBoundingBoxPositions(
            height,
            width,
            face.BoundingBox,
            detectFacesResponse.OrientationCorrection);

        Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
        Console.WriteLine($"The detected face is estimated to be between
{face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
    });
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

/// <summary>
/// Display the bounding box information for an image.
/// </summary>
/// <param name="imageHeight">The height of the image.</param>
/// <param name="imageWidth">The width of the image.</param>
/// <param name="box">The bounding box for a face found within the image.</
param>
/// <param name="rotation">The rotation of the face's bounding box.</param>
public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
BoundingBox box, string rotation)
{
    float left;
    float top;

    if (rotation == null)
    {
        Console.WriteLine("No estimated orientation. Check Exif data.");
        return;
    }

    // Calculate face position based on image orientation.
    switch (rotation)
    {
        case "ROTATE_0":

```

```

        left = imageWidth * box.Left;
        top = imageHeight * box.Top;
        break;
    case "ROTATE_90":
        left = imageHeight * (1 - (box.Top + box.Height));
        top = imageWidth * box.Left;
        break;
    case "ROTATE_180":
        left = imageWidth - (imageWidth * (box.Left + box.Width));
        top = imageHeight * (1 - (box.Top + box.Height));
        break;
    case "ROTATE_270":
        left = imageHeight * box.Top;
        top = imageWidth * (1 - box.Left - box.Width);
        break;
    default:
        Console.WriteLine("No estimated orientation information. Check
Exif data.");
        return;
    }

    // Display face location information.
    Console.WriteLine($"Left: {left}");
    Console.WriteLine($"Top: {top}");
    Console.WriteLine($"Face Width: {imageWidth * box.Width}");
    Console.WriteLine($"Face Height: {imageHeight * box.Height}");
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DetectFaces](#) à la section Référence des AWS SDK pour .NET API.

DetectLabels

L'exemple de code suivant montre comment utiliser `DetectLabels`.

Pour plus d'informations, veuillez consulter [Détection des étiquettes dans une image](#).

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectLabels
{
    public static async Task Main()
    {
        string photo = "del_river_02092020_01.jpg"; // "input.jpg";
        string bucket = "amzn-s3-demo-bucket"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectLabelsRequest = new DetectLabelsRequest
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F,
        };

        try
        {
```

```

        DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectLabelsRequest);
        Console.WriteLine("Detected labels for " + photo);
        foreach (Label label in detectLabelsResponse.Labels)
        {
            Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
}

```

Détectez les étiquettes dans un fichier image stocké sur votre ordinateur.

```

using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored locally.
/// </summary>
public class DetectLabelsLocalFile
{
    public static async Task Main()
    {
        string photo = "input.jpg";

        var image = new Amazon.Rekognition.Model.Image();
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
            byte[] data = null;
            data = new byte[fs.Length];

```



```

        fs.Read(data, 0, (int)fs.Length);
        image.Bytes = new MemoryStream(data);
    }
    catch (Exception)
    {
        Console.WriteLine("Failed to load file " + photo);
        return;
    }

    var rekognitionClient = new AmazonRekognitionClient();

    var detectLabelsRequest = new DetectLabelsRequest
    {
        Image = image,
        MaxLabels = 10,
        MinConfidence = 77F,
    };

    try
    {
        DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectLabelsRequest);
        Console.WriteLine($"Detected labels for {photo}");
        foreach (Label label in detectLabelsResponse.Labels)
        {
            Console.WriteLine($"{label.Name}: {label.Confidence}");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DetectLabels](#) à la section Référence des AWS SDK pour .NET API.

DetectModerationLabels

L'exemple de code suivant montre comment utiliser `DetectModerationLabels`.

Pour plus d'informations, veuillez consulter [Détecter des images inappropriées](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect unsafe content in a
/// JPEG or PNG format image.
/// </summary>
public class DetectModerationLabels
{
    public static async Task Main(string[] args)
    {
        string photo = "input.jpg";
        string bucket = "amzn-s3-demo-bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectModerationLabelsRequest = new DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
            MinConfidence = 60F,
        };

        try
```

```
{
    var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
    Console.WriteLine("Detected labels for " + photo);
    foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
    {
        Console.WriteLine($"Label: {label.Name}");
        Console.WriteLine($"Confidence: {label.Confidence}");
        Console.WriteLine($"Parent: {label.ParentName}");
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectModerationLabels](#) à la section Référence des AWS SDK pour .NET API.

DetectText

L'exemple de code suivant montre comment utiliser `DetectText`.

Pour plus d'informations, consultez [Détection de texte dans une image](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
/// <summary>
/// Uses the Amazon Rekognition Service to detect text in an image. The
/// example was created using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class DetectText
{
    public static async Task Main()
    {
        string photo = "Dad_photographer.jpg"; // "input.jpg";
        string bucket = "amzn-s3-demo-bucket"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
        };

        try
        {
            DetectTextResponse detectTextResponse = await
rekognitionClient.DetectTextAsync(detectTextRequest);
            Console.WriteLine($"Detected lines and words for {photo}");
            detectTextResponse.TextDetections.ForEach(text =>
            {
                Console.WriteLine($"Detected: {text.DetectedText}");
                Console.WriteLine($"Confidence: {text.Confidence}");
                Console.WriteLine($"Id : {text.Id}");
                Console.WriteLine($"Parent Id: {text.ParentId}");
                Console.WriteLine($"Type: {text.Type}");
            });
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

```
    }  
  }  
}
```

- Pour plus de détails sur l'API, reportez-vous [DetectText](#) à la section Référence des AWS SDK pour .NET API.

GetCelebrityInfo

L'exemple de code suivant montre comment utiliser `GetCelebrityInfo`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Rekognition;  
using Amazon.Rekognition.Model;  
  
/// <summary>  
/// Shows how to use Amazon Rekognition to retrieve information about the  
/// celebrity identified by the supplied celebrity Id.  
/// </summary>  
public class CelebrityInfo  
{  
    public static async Task Main()  
    {  
        string celebId = "nnnnnnnnn";  
  
        var rekognitionClient = new AmazonRekognitionClient();  
  
        var celebrityInfoRequest = new GetCelebrityInfoRequest  
        {  
            Id = celebId,  

```

```
};

Console.WriteLine($"Getting information for celebrity: {celebId}");

var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

// Display celebrity information.
Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
Console.WriteLine("Further information (if available):");
celebrityInfoResponse.Urls.ForEach(url =>
{
    Console.WriteLine(url);
});
}
```

- Pour plus de détails sur l'API, reportez-vous [GetCelebrityInfo](#) à la section Référence des AWS SDK pour .NET API.

IndexFaces

L'exemple de code suivant montre comment utiliser `IndexFaces`.

Pour plus d'informations, veuillez consulter [Ajouter des visages à une collection](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "amzn-s3-demo-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var indexFacesRequest = new IndexFacesRequest
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<string>() { "ALL" },
        };

        IndexFacesResponse indexFacesResponse = await
        rekognitionClient.IndexFacesAsync(indexFacesRequest);

        Console.WriteLine($"{photo} added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
        {
            Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [IndexFaces](#) à la section Référence des AWS SDK pour .NET API.

ListCollections

L'exemple de code suivant montre comment utiliser `ListCollections`.

Pour plus d'informations, consultez [Créer une liste de collections](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses Amazon Rekognition to list the collection IDs in the
/// current account.
/// </summary>
public class ListCollections
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        var listCollectionsRequest = new ListCollectionsRequest
        {
            MaxResults = limit,
        };
    }
}
```



```
var listCollectionsResponse = new ListCollectionsResponse();

do
{
    if (listCollectionsResponse is not null)
    {
        listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
    }

    listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

    listCollectionsResponse.CollectionIds.ForEach(id =>
    {
        Console.WriteLine(id);
    });
}
while (listCollectionsResponse.NextToken is not null);
}
```

- Pour plus de détails sur l'API, reportez-vous [ListCollections](#) à la section Référence des AWS SDK pour .NET API.

ListFaces

L'exemple de code suivant montre comment utiliser `ListFaces`.

Pour plus d'informations, consultez [Répertoire de visages d'une collection](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
```

```
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to retrieve the list of faces
/// stored in a collection.
/// </summary>
public class ListFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";

        var rekognitionClient = new AmazonRekognitionClient();

        var listFacesResponse = new ListFacesResponse();
        Console.WriteLine($"Faces in collection {collectionId}");

        var listFacesRequest = new ListFacesRequest
        {
            CollectionId = collectionId,
            MaxResults = 1,
        };

        do
        {
            listFacesResponse = await
rekognitionClient.ListFacesAsync(listFacesRequest);
            listFacesResponse.Faces.ForEach(face =>
            {
                Console.WriteLine(face.FaceId);
            });

            listFacesRequest.NextToken = listFacesResponse.NextToken;
        }
        while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListFaces](#) à la section Référence des AWS SDK pour .NET API.

RecognizeCelebrities

L'exemple de code suivant montre comment utiliser `RecognizeCelebrities`.

Pour plus d'informations, consultez [Reconnaissance de célébrités dans une image](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Shows how to use Amazon Rekognition to identify celebrities in a photo.
/// </summary>
public class CelebritiesInImage
{
    public static async Task Main(string[] args)
    {
        string photo = "moviestars.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

        var img = new Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
                FileAccess.Read);
```

```

        data = new byte[fs.Length];
        fs.Read(data, 0, (int)fs.Length);
    }
    catch (Exception)
    {
        Console.WriteLine($"Failed to load file {photo}");
        return;
    }

    img.Bytes = new MemoryStream(data);
    recognizeCelebritiesRequest.Image = img;

    Console.WriteLine($"Looking for celebrities in image {photo}\n");

    var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

    Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
    recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
    {
        Console.WriteLine($"Celebrity recognized: {celeb.Name}");
        Console.WriteLine($"Celebrity ID: {celeb.Id}");
        BoundingBox boundingBox = celeb.Face.BoundingBox;
        Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
        Console.WriteLine("Further information (if available):");
        celeb.UrlsWithMetadata.ForEach(url =>
        {
            Console.WriteLine(url);
        });
    });
});

Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [RecognizeCelebrities](#) à la section Référence des AWS SDK pour .NET API.

SearchFaces

L'exemple de code suivant montre comment utiliser SearchFaces.

Pour plus d'informations, veuillez consulter [Recherche d'un visage \(identification faciale\)](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to find faces in an image that
/// match the face Id provided in the method request.
/// </summary>
public class SearchFacesMatchingId
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx";

        var rekognitionClient = new AmazonRekognitionClient();

        // Search collection for faces matching the face id.
        var searchFacesRequest = new SearchFacesRequest
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
            MaxFaces = 2,
        };

        SearchFacesResponse searchFacesResponse = await
            rekognitionClient.SearchFacesAsync(searchFacesRequest);
    }
}
```

```
        Console.WriteLine("Face matching faceId " + faceId);

        Console.WriteLine("Matche(s): ");
        searchFacesResponse.FaceMatches.ForEach(face =>
        {
            Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
{face.Similarity}");
        });
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [SearchFaces](#) à la section Référence des AWS SDK pour .NET API.

SearchFacesByImage

L'exemple de code suivant montre comment utiliser `SearchFacesByImage`.

Pour plus d'informations, voir [Recherche d'un visage \(image\)](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to search for images matching those
/// in a collection.
/// </summary>
public class SearchFacesMatchingImage
```

```

{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string bucket = "amzn-s3-demo-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        var image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var searchFacesByImageRequest = new SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
            Image = image,
            FaceMatchThreshold = 70F,
            MaxFaces = 2,
        };

        SearchFacesByImageResponse searchFacesByImageResponse = await
        rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

        Console.WriteLine("Faces matching largest face in image from " + photo);
        searchFacesByImageResponse.FaceMatches.ForEach(face =>
        {
            Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
        });
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [SearchFacesByImage](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour .NET

Montre comment utiliser l'API Java Amazon Rekognition afin de créer une application qui, avec Amazon Rekognition, permet d'identifier des objets par catégorie dans des images stockées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemples d'enregistrement de domaine Route 53 à l'aide de SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de l'enregistrement de domaine AWS SDK pour .NET avec Route 53.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour Enregistrement de domaine Route 53

L'exemple de code suivant montre comment commencer à utiliser l'enregistrement de domaine Route 53.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
public static class HelloRoute53Domains
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the Amazon Route 53 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRoute53Domains>()
            ).Build();

        // Now the client is available for injection.
        var route53Client =
            host.Services.GetRequiredService<IAmazonRoute53Domains>();

        // You can use await and any of the async methods to get a response.
        var response = await route53Client.ListPricesAsync(new ListPricesRequest
        { Tld = "com" });
        Console.WriteLine($"Hello Amazon Route 53 Domains! Following are prices
        for .com domain operations:");
        var comPrices = response.Prices.FirstOrDefault();
        if (comPrices != null)
        {
            Console.WriteLine($"Registration: {comPrices.RegistrationPrice?.Price}
            {comPrices.RegistrationPrice?.Currency}");
            Console.WriteLine($"Renewal: {comPrices.RenewalPrice?.Price}
            {comPrices.RenewalPrice?.Currency}");
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListPrices](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Répertorier les domaines actuels et les opérations effectuées au cours de l'année écoulée.
- Afficher la facturation de l'année écoulée et les prix des types de domaines.
- Obtenir des suggestions de domaines.
- Vérifier la disponibilité et la transférabilité du domaine.
- Éventuellement, demander l'enregistrement d'un domaine.
- Obtenir des informations sur une opération.
- Éventuellement, obtenir des informations sur un domaine.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
public static class Route53DomainScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. List current domains.
        2. List operations in the past year.
        3. View billing for the account in the past year.
```

```

    4. View prices for domain types.
    5. Get domain suggestions.
    6. Check domain availability.
    7. Check domain transferability.
    8. Optionally, request a domain registration.
    9. Get an operation detail.
    10. Optionally, get a domain detail.
*/

private static Route53Wrapper _route53Wrapper = null!;
private static IConfiguration _configuration = null!;

static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonRoute53Domains>()
                .AddTransient<Route53Wrapper>()
        )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    var logger = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
    }).CreateLogger(typeof(Route53DomainScenario));

    _route53Wrapper = host.Services.GetRequiredService<Route53Wrapper>();

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Amazon Route 53 domains example
scenario.");

```

```

        Console.WriteLine(new string('-', 80));

        try
        {
            await ListDomains();
            await ListOperations();
            await ListBillingRecords();
            await ListPrices();
            await ListDomainSuggestions();
            await CheckDomainAvailability();
            await CheckDomainTransferability();
            var operationId = await RequestDomainRegistration();
            await GetOperationalDetail(operationId);
            await GetDomainDetails();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The Amazon Route 53 domains example scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List account registered domains.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDomains()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List account domains.");
        var domains = await _route53Wrapper.ListDomains();
        for (int i = 0; i < domains.Count; i++)
        {
            Console.WriteLine($"  {i + 1}. {domains[i].DomainName}");
        }

        if (!domains.Any())
        {
            Console.WriteLine($"  No domains found in this account.");
        }
    }

```

```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain operations in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListOperations()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List account domain operations in the past year.");
        var operations = await _route53Wrapper.ListOperations(
            DateTime.Today.AddYears(-1));
        for (int i = 0; i < operations.Count; i++)
        {
            Console.WriteLine($"\\tOperation Id: {operations[i].OperationId}");
            Console.WriteLine($"\\tStatus: {operations[i].Status}");
            Console.WriteLine($"\\tDate: {operations[i].SubmittedDate}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List billing in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListBillingRecords()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. View billing for the account in the past year.");
        var billingRecords = await _route53Wrapper.ViewBilling(
            DateTime.Today.AddYears(-1),
            DateTime.Today);
        for (int i = 0; i < billingRecords.Count; i++)
        {
            Console.WriteLine($"\\tBill Date:
{billingRecords[i].BillDate.ToShortDateString()}");
            Console.WriteLine($"\\tOperation: {billingRecords[i].Operation}");
            Console.WriteLine($"\\tPrice: {billingRecords[i].Price}");
        }
        if (!billingRecords.Any())
        {

```

```

        Console.WriteLine("\tNo billing records found in this account for the
past year.");
    }
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List prices for a few domain types.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListPrices()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"4. View prices for domain types.");
    var domainTypes = new List<string> { "net", "com", "org", "co" };

    var prices = await _route53Wrapper.ListPrices(domainTypes);
    foreach (var pr in prices)
    {
        Console.WriteLine($"    \tName: {pr.Name}");
        Console.WriteLine($"    \tRegistration: {pr.RegistrationPrice?.Price}
{pr.RegistrationPrice?.Currency}");
        Console.WriteLine($"    \tRenewal: {pr.RenewalPrice?.Price}
{pr.RenewalPrice?.Currency}");
        Console.WriteLine($"    \tTransfer: {pr.TransferPrice?.Price}
{pr.TransferPrice?.Currency}");
        Console.WriteLine($"    \tChange Ownership: {pr.ChangeOwnershipPrice?.Price}
{pr.ChangeOwnershipPrice?.Currency}");
        Console.WriteLine($"    \tRestoration: {pr.RestorationPrice?.Price}
{pr.RestorationPrice?.Currency}");
        Console.WriteLine();
    }
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List domain suggestions for a domain name.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDomainSuggestions()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"5. Get domain suggestions.");
    string? domainName = null;

```

```

        while (domainName == null || string.IsNullOrEmpty(domainName))
        {
            Console.WriteLine($"Enter a domain name to get available domain
suggestions.");
            domainName = Console.ReadLine();
        }

        var suggestions = await _route53Wrapper.GetDomainSuggestions(domainName,
true, 5);
        foreach (var suggestion in suggestions)
        {
            Console.WriteLine($"\\tSuggestion Name: {suggestion.DomainName}");
            Console.WriteLine($"\\tAvailability: {suggestion.Availability}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check availability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainAvailability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Check domain availability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrEmpty(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain availability.");
            domainName = Console.ReadLine();
        }

        var availability = await
_route53Wrapper.CheckDomainAvailability(domainName);
        Console.WriteLine($"\\tAvailability: {availability}");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainTransferability()
    {

```



```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Check domain transferability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain
transferability.");
            domainName = Console.ReadLine();
        }

        var transferability = await
_route53Wrapper.CheckDomainTransferability(domainName);
        Console.WriteLine($"Transferability: {transferability}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> RequestDomainRegistration()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Optionally, request a domain registration.");

        Console.WriteLine($"Note: This example uses domain request settings in
settings.json.");
        Console.WriteLine($"To change the domain registration settings, set the
values in that file.");
        Console.WriteLine($"Remember, registering an actual domain will incur an
account billing cost.");
        Console.WriteLine($"Would you like to begin a domain registration? (y/
n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase))
        {
            string domainName = _configuration["DomainName"];
            ContactDetail contact = new ContactDetail();
            contact.CountryCode =
CountryCode.FindValue(_configuration["Contact:CountryCode"]);
            contact.ContactType =
ContactType.FindValue(_configuration["Contact:ContactType"]);

```

```

        _configuration.GetSection("Contact").Bind(contact);

        var operationId = await _route53Wrapper.RegisterDomain(
            domainName,
            Convert.ToBoolean(_configuration["AutoRenew"]),
            Convert.ToInt32(_configuration["DurationInYears"]),
            contact);
        if (operationId != null)
        {
            Console.WriteLine(
                $"\\tRegistration requested. Operation Id: {operationId}");
        }

        return operationId;
    }

    Console.WriteLine(new string('-', 80));
    return null;
}

/// <summary>
/// Get details for an operation.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetOperationalDetail(string? operationId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Get an operation detail.");

    var operationDetails =
        await _route53Wrapper.GetOperationDetail(operationId);

    Console.WriteLine(operationDetails);

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Optionally, get details for a registered domain.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string?> GetDomainDetails()
{

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Get details on a domain.");

        Console.WriteLine($"\\tNote: you must have a registered domain to get
details.");
        Console.WriteLine($"\\tWould you like to get domain details? (y/n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase))
        {
            string? domainName = null;
            while (domainName == null)
            {
                Console.WriteLine($"\\tEnter a domain name to get details.");
                domainName = Console.ReadLine();
            }

            var domainDetails = await _route53Wrapper.GetDomainDetail(domainName);
            Console.WriteLine(domainDetails);
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }
}

```

Méthodes d'encapsulation utilisées par le scénario pour les actions d'enregistrement de domaine Route 53.

```

public class Route53Wrapper
{
    private readonly IAmazonRoute53Domains _amazonRoute53Domains;
    private readonly ILogger<Route53Wrapper> _logger;
    public Route53Wrapper(IAmazonRoute53Domains amazonRoute53Domains,
ILogger<Route53Wrapper> logger)
    {
        _amazonRoute53Domains = amazonRoute53Domains;
        _logger = logger;
    }
}

```

```
/// <summary>
/// List prices for domain type operations.
/// </summary>
/// <param name="domainTypes">Domain types to include in the results.</param>
/// <returns>The list of domain prices.</returns>
public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
{
    var results = new List<DomainPrice>();
    var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
ListPricesRequest());
    // Get the entire list using the paginator.
    await foreach (var prices in paginatePrices.Prices)
    {
        results.Add(prices);
    }
    return results.Where(p => domainTypes.Contains(p.Name)).ToList();
}

/// <summary>
/// Check the availability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for availability.</param>
/// <returns>An availability result string.</returns>
public async Task<string> CheckDomainAvailability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
        new CheckDomainAvailabilityRequest
        {
            DomainName = domain
        }
    );
    return result.Availability.Value;
}

/// <summary>
/// Check the transferability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for transferability.</param>
/// <returns>A transferability result string.</returns>
public async Task<string> CheckDomainTransferability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
```

```

        new CheckDomainTransferabilityRequest
        {
            DomainName = domain
        }
    );
    return result.Transferability.Transferable.Value;
}

/// <summary>
/// Get a list of suggestions for a given domain.
/// </summary>
/// <param name="domain">The domain to check for suggestions.</param>
/// <param name="onlyAvailable">If true, only returns available domains.</param>
/// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
/// <returns>A collection of domain suggestions.</returns>
public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
{
    var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
        new GetDomainSuggestionsRequest
        {
            DomainName = domain,
            OnlyAvailable = onlyAvailable,
            SuggestionCount = suggestionCount
        }
    );
    return result.SuggestionsList;
}

/// <summary>
/// Get details for a domain action operation.
/// </summary>
/// <param name="operationId">The operational Id.</param>
/// <returns>A string describing the operational details.</returns>
public async Task<string> GetOperationDetail(string? operationId)
{
    if (operationId == null)
        return "Unable to get operational details because ID is null.";
    try
    {
        var operationDetails =

```

```

        await _amazonRoute53Domains.GetOperationDetailAsync(
            new GetOperationDetailRequest
            {
                OperationId = operationId
            }
        );

        var details = $"\\tOperation {operationId}:\\n" +
            $"\\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}\\n" +
            $"\\tMessage is {operationDetails.Message}\\n" +
            $"\\tStatus is {operationDetails.Status}\\n";

        return details;
    }
    catch (AmazonRoute53DomainsException ex)
    {
        return $"Unable to get operation details. Here's why: {ex.Message}.";
    }
}

/// <summary>
/// Initiate a domain registration request.
/// </summary>
/// <param name="contact">Contact details.</param>
/// <param name="domainName">The domain name to register.</param>
/// <param name="autoRenew">True if the domain should automatically renew.</
param>
/// <param name="duration">The duration in years for the domain registration.</
param>
/// <returns>The operation Id.</returns>
public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
duration, ContactDetail contact)
{
    // This example uses the same contact information for admin, registrant, and
    tech contacts.
    try
    {
        var result = await _amazonRoute53Domains.RegisterDomainAsync(
            new RegisterDomainRequest()
            {
                AdminContact = contact,
                RegistrantContact = contact,

```

```

        TechContact = contact,
        DomainName = domainName,
        AutoRenew = autoRenew,
        DurationInYears = duration,
        PrivacyProtectAdminContact = false,
        PrivacyProtectRegistrantContact = false,
        PrivacyProtectTechContact = false
    }
    );
    return result.OperationId;
}
catch (InvalidInputException)
{
    _logger.LogInformation($"Unable to request registration for domain
{domainName}");
    return null;
}
}

/// <summary>
/// View billing records for the account between a start and end date.
/// </summary>
/// <param name="startDate">The start date for billing results.</param>
/// <param name="endDate">The end date for billing results.</param>
/// <returns>A collection of billing records.</returns>
public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
{
    var results = new List<BillingRecord>();
    var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
        new ViewBillingRequest()
        {
            Start = startDate,
            End = endDate
        });

    // Get the entire list using the paginator.
    await foreach (var billingRecords in paginateBilling.BillingRecords)
    {
        results.Add(billingRecords);
    }
    return results;
}

```

```
/// <summary>
/// List the domains for the account.
/// </summary>
/// <returns>A collection of domain summary records.</returns>
public async Task<List<DomainSummary>> ListDomains()
{
    var results = new List<DomainSummary>();
    var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
        new ListDomainsRequest());

    // Get the entire list using the paginator.
    await foreach (var domain in paginateDomains.Domains)
    {
        results.Add(domain);
    }
    return results;
}

/// <summary>
/// List operations for the account that are submitted after a specified date.
/// </summary>
/// <returns>A collection of operation summary records.</returns>
public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
{
    var results = new List<OperationSummary>();
    var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
        new ListOperationsRequest()
        {
            SubmittedSince = submittedSince
        });

    // Get the entire list using the paginator.
    await foreach (var operations in paginateOperations.Operations)
    {
        results.Add(operations);
    }
    return results;
}
```



```
/// <summary>
/// Get details for a domain.
/// </summary>
/// <returns>A string with detail information about the domain.</returns>
public async Task<string> GetDomainDetail(string domainName)
{
    try
    {
        var result = await _amazonRoute53Domains.GetDomainDetailAsync(
            new GetDomainDetailRequest()
            {
                DomainName = domainName
            });
        var details = $"\\tDomain {domainName}:\\n" +
            $"\\tCreated on {result.CreationDate.ToShortDateString()}.\\n" +
            $"\\tAdmin contact is {result.AdminContact.Email}.\\n" +
            $"\\tAuto-renew is {result.AutoRenew}.\\n";

        return details;
    }
    catch (InvalidInputException)
    {
        return $"Domain {domainName} was not found in your account.";
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .

- [CheckDomainAvailability](#)
- [CheckDomainTransferability](#)
- [GetDomainDetail](#)
- [GetDomainSuggestions](#)
- [GetOperationDetail](#)
- [ListDomains](#)
- [ListOperations](#)
- [ListPrices](#)

- [RegisterDomain](#)
- [ViewBilling](#)

Actions

CheckDomainAvailability

L'exemple de code suivant montre comment utiliser `CheckDomainAvailability`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Check the availability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for availability.</param>
/// <returns>An availability result string.</returns>
public async Task<string> CheckDomainAvailability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
        new CheckDomainAvailabilityRequest
        {
            DomainName = domain
        }
    );
    return result.Availability.Value;
}
```

- Pour plus de détails sur l'API, reportez-vous [CheckDomainAvailability](#) à la section Référence des AWS SDK pour .NET API.

CheckDomainTransferability

L'exemple de code suivant montre comment utiliser `CheckDomainTransferability`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Check the transferability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for transferability.</param>
/// <returns>A transferability result string.</returns>
public async Task<string> CheckDomainTransferability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
        new CheckDomainTransferabilityRequest
        {
            DomainName = domain
        }
    );
    return result.Transferability.Transferable.Value;
}
```

- Pour plus de détails sur l'API, reportez-vous [CheckDomainTransferability](#) à la section Référence des AWS SDK pour .NET API.

GetDomainDetail

L'exemple de code suivant montre comment utiliser `GetDomainDetail`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get details for a domain.
/// </summary>
/// <returns>A string with detail information about the domain.</returns>
public async Task<string> GetDomainDetail(string domainName)
{
    try
    {
        var result = await _amazonRoute53Domains.GetDomainDetailAsync(
            new GetDomainDetailRequest()
            {
                DomainName = domainName
            });
        var details = $"\\tDomain {domainName}:\\n" +
            $"\\tCreated on {result.CreationDate.ToShortDateString()}.\\n" +
            $"\\tAdmin contact is {result.AdminContact.Email}.\\n" +
            $"\\tAuto-renew is {result.AutoRenew}.\\n";

        return details;
    }
    catch (InvalidInputException)
    {
        return $"Domain {domainName} was not found in your account.";
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [GetDomainDetail](#) à la section Référence des AWS SDK pour .NET API.

GetDomainSuggestions

L'exemple de code suivant montre comment utiliser `GetDomainSuggestions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get a list of suggestions for a given domain.
/// </summary>
/// <param name="domain">The domain to check for suggestions.</param>
/// <param name="onlyAvailable">If true, only returns available domains.</param>
/// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
/// <returns>A collection of domain suggestions.</returns>
public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
{
    var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
        new GetDomainSuggestionsRequest
        {
            DomainName = domain,
            OnlyAvailable = onlyAvailable,
            SuggestionCount = suggestionCount
        }
    );
    return result.SuggestionsList;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetDomainSuggestions](#) à la section Référence des AWS SDK pour .NET API.

GetOperationDetail

L'exemple de code suivant montre comment utiliser `GetOperationDetail`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get details for a domain action operation.
/// </summary>
/// <param name="operationId">The operational Id.</param>
/// <returns>A string describing the operational details.</returns>
public async Task<string> GetOperationDetail(string? operationId)
{
    if (operationId == null)
        return "Unable to get operational details because ID is null.";
    try
    {
        var operationDetails =
            await _amazonRoute53Domains.GetOperationDetailAsync(
                new GetOperationDetailRequest
                {
                    OperationId = operationId
                }
            );

        var details = $"\\tOperation {operationId}:\\n" +
            $"\\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}\\n" +
            $"\\tMessage is {operationDetails.Message}\\n" +
            $"\\tStatus is {operationDetails.Status}\\n";

        return details;
    }
    catch (AmazonRoute53DomainsException ex)
    {
        return $"Unable to get operation details. Here's why: {ex.Message}.";
    }
}
```

```
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [GetOperationDetail](#) à la section Référence des AWS SDK pour .NET API.

ListDomains

L'exemple de code suivant montre comment utiliser `ListDomains`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// List the domains for the account.  
/// </summary>  
/// <returns>A collection of domain summary records.</returns>  
public async Task<List<DomainSummary>> ListDomains()  
{  
    var results = new List<DomainSummary>();  
    var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(  
        new ListDomainsRequest());  
  
    // Get the entire list using the paginator.  
    await foreach (var domain in paginateDomains.Domains)  
    {  
        results.Add(domain);  
    }  
    return results;  
}
```

- Pour plus de détails sur l'API, reportez-vous [ListDomains](#) à la section Référence des AWS SDK pour .NET API.

ListOperations

L'exemple de code suivant montre comment utiliser `ListOperations`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List operations for the account that are submitted after a specified date.
/// </summary>
/// <returns>A collection of operation summary records.</returns>
public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
{
    var results = new List<OperationSummary>();
    var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
        new ListOperationsRequest()
        {
            SubmittedSince = submittedSince
        });

    // Get the entire list using the paginator.
    await foreach (var operations in paginateOperations.Operations)
    {
        results.Add(operations);
    }
    return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListOperations](#) à la section Référence des AWS SDK pour .NET API.

ListPrices

L'exemple de code suivant montre comment utiliser `ListPrices`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List prices for domain type operations.
/// </summary>
/// <param name="domainTypes">Domain types to include in the results.</param>
/// <returns>The list of domain prices.</returns>
public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
{
    var results = new List<DomainPrice>();
    var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
ListPricesRequest());
    // Get the entire list using the paginator.
    await foreach (var prices in paginatePrices.Prices)
    {
        results.Add(prices);
    }
    return results.Where(p => domainTypes.Contains(p.Name)).ToList();
}
```

- Pour plus de détails sur l'API, reportez-vous [ListPrices](#) à la section Référence des AWS SDK pour .NET API.

RegisterDomain

L'exemple de code suivant montre comment utiliser `RegisterDomain`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Initiate a domain registration request.
/// </summary>
/// <param name="contact">Contact details.</param>
/// <param name="domainName">The domain name to register.</param>
/// <param name="autoRenew">True if the domain should automatically renew.</
param>
/// <param name="duration">The duration in years for the domain registration.</
param>
/// <returns>The operation Id.</returns>
public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
duration, ContactDetail contact)
{
    // This example uses the same contact information for admin, registrant, and
    tech contacts.
    try
    {
        var result = await _amazonRoute53Domains.RegisterDomainAsync(
            new RegisterDomainRequest()
            {
                AdminContact = contact,
                RegistrantContact = contact,
                TechContact = contact,
                DomainName = domainName,
                AutoRenew = autoRenew,
                DurationInYears = duration,
                PrivacyProtectAdminContact = false,
                PrivacyProtectRegistrantContact = false,
                PrivacyProtectTechContact = false
            }
        );
        return result.OperationId;
    }
}
```

```
        catch (InvalidInputException)
        {
            _logger.LogInformation($"Unable to request registration for domain {domainName}");
            return null;
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [RegisterDomain](#) à la section Référence des AWS SDK pour .NET API.

ViewBilling

L'exemple de code suivant montre comment utiliser `ViewBilling`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// View billing records for the account between a start and end date.
/// </summary>
/// <param name="startDate">The start date for billing results.</param>
/// <param name="endDate">The end date for billing results.</param>
/// <returns>A collection of billing records.</returns>
public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
{
    var results = new List<BillingRecord>();
    var paginateBilling = _amazonRoute53Domains.Paginateors.ViewBilling(
        new ViewBillingRequest()
        {
            Start = startDate,
            End = endDate
        });
}
```

```
// Get the entire list using the paginator.
await foreach (var billingRecords in paginateBilling.BillingRecords)
{
    results.Add(billingRecords);
}
return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [ViewBilling](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'utilisation d'Amazon S3 SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon S3.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)

Actions

CopyObject

L'exemple de code suivant montre comment utiliser `CopyObject`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Copiez un objet à l'aide d'une demande conditionnelle.

```

    /// <summary>
    /// Copies an object from one Amazon S3 bucket to another with a conditional
    request.
    /// </summary>
    /// <param name="sourceKey">The key of the source object to copy.</param>
    /// <param name="destKey">The key of the destination object.</param>
    /// <param name="sourceBucket">The source bucket of the object.</param>
    /// <param name="destBucket">The destination bucket of the object.</param>
    /// <param name="conditionType">The type of condition to apply, e.g.
    'CopySourceIfMatch', 'CopySourceIfNoneMatch', 'CopySourceIfModifiedSince',
    'CopySourceIfUnmodifiedSince'.</param>
    /// <param name="conditionDateValue">The value to use for the condition for
    dates.</param>
    /// <param name="etagConditionalValue">The value to use for the condition for
    etags.</param>
    /// <returns>True if the conditional copy is successful, False otherwise.</
    returns>
    public async Task<bool> CopyObjectConditional(string sourceKey, string destKey,
        string sourceBucket, string destBucket,
        S3ConditionType conditionType, DateTime? conditionDateValue = null, string?
        etagConditionalValue = null)
    {
        try
        {
            var copyObjectRequest = new CopyObjectRequest
            {
                DestinationBucket = destBucket,
                DestinationKey = destKey,
                SourceBucket = sourceBucket,
                SourceKey = sourceKey
            };

            switch (conditionType)

```

```

        {
            case S3ConditionType.IfMatch:
                copyObjectRequest.ETagToMatch = etagConditionalValue;
                break;
            case S3ConditionType.IfNoneMatch:
                copyObjectRequest.ETagToNotMatch = etagConditionalValue;
                break;
            case S3ConditionType.IfModifiedSince:
                copyObjectRequest.ModifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
                break;
            case S3ConditionType.IfUnmodifiedSince:
                copyObjectRequest.UnmodifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
                break;
            default:
                throw new ArgumentOutOfRangeException(nameof(conditionType),
conditionType, null);
        }

        await _amazonS3.CopyObjectAsync(copyObjectRequest);
        _logger.LogInformation($"Conditional copy successful for key {destKey}
in bucket {destBucket}.");
        return true;
    }
    catch (AmazonS3Exception e)
    {
        if (e.ErrorCode == "PreconditionFailed")
        {
            _logger.LogError("Conditional copy failed: Precondition failed");
        }
        else if (e.ErrorCode == "304")
        {
            _logger.LogError("Conditional copy failed: Object not modified");
        }
        else
        {
            _logger.LogError($"Unexpected error: {e.ErrorCode}");
            throw;
        }
        return false;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CopyObject](#) à la section Référence des AWS SDK pour .NET API.

CreateBucket

L'exemple de code suivant montre comment utiliser `CreateBucket`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez un compartiment avec le verrouillage d'objet activé.

```
/// <summary>
/// Create a new Amazon S3 bucket with object lock actions.
/// </summary>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
enableObjectLock)
{
    Console.WriteLine($"\\tCreating bucket {bucketName} with object lock
{enableObjectLock}.");
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
            ObjectLockEnabledForBucket = enableObjectLock,
        };

        var response = await _amazonS3.PutBucketAsync(request);
    }
}
```

```

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error creating bucket: '{ex.Message}'");
        return false;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateBucket](#) à la section Référence des AWS SDK pour .NET API.

DeleteBucketCors

L'exemple de code suivant montre comment utiliser `DeleteBucketCors`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Deletes a CORS configuration from an Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to delete the CORS configuration from the bucket.</param>
private static async Task DeleteCORSConfigurationAsync(AmazonS3Client
client)
{
    DeleteCORSConfigurationRequest request = new
DeleteCORSConfigurationRequest()
    {
        BucketName = BucketName,
    };
    await client.DeleteCORSConfigurationAsync(request);
}

```


- Pour plus de détails sur l'API, reportez-vous [DeleteBucketCors](#) à la section Référence des AWS SDK pour .NET API.

DeleteBucketLifecycle

L'exemple de code suivant montre comment utiliser `DeleteBucketLifecycle`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// This method removes the Lifecycle configuration from the named
/// S3 bucket.
/// </summary>
/// <param name="client">The S3 client object used to call
/// the RemoveLifecycleConfigAsync method.</param>
/// <param name="bucketName">A string representing the name of the
/// S3 bucket from which the configuration will be removed.</param>
public static async Task RemoveLifecycleConfigAsync(IAmazonS3 client, string
bucketName)
{
    var request = new DeleteLifecycleConfigurationRequest()
    {
        BucketName = bucketName,
    };
    await client.DeleteLifecycleConfigurationAsync(request);
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteBucketLifecycle](#) à la section Référence des AWS SDK pour .NET API.

DeleteObject

L'exemple de code suivant montre comment utiliser `DeleteObject`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Supprimez un objet dans un compartiment S3 non soumis à la gestion des versions.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete an object from a non-versioned Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteObject
{
    /// <summary>
    /// The Main method initializes the necessary variables and then calls
    /// the DeleteObjectNonVersionedBucketAsync method to delete the object
    /// named by the keyName parameter.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "amzn-s3-demo-bucket";
        const string keyName = "testfile.txt";

        // If the Amazon S3 bucket is located in an AWS Region other than the
        // Region of the default account, define the AWS Region for the
        // Amazon S3 bucket in your call to the AmazonS3Client constructor.
        // For example RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
        await DeleteObjectNonVersionedBucketAsync(client, bucketName, keyName);
    }
}
```

```

    /// <summary>
    /// The DeleteObjectNonVersionedBucketAsync takes care of deleting the
    /// desired object from the named bucket.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client used to delete
    /// an object from an Amazon S3 bucket.</param>
    /// <param name="bucketName">The name of the bucket from which the
    /// object will be deleted.</param>
    /// <param name="keyName">The name of the object to delete.</param>
    public static async Task DeleteObjectNonVersionedBucketAsync(IAmazonS3
client, string bucketName, string keyName)
    {
        try
        {
            var deleteObjectRequest = new DeleteObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
            };

            Console.WriteLine($"Deleting object: {keyName}");
            await client.DeleteObjectAsync(deleteObjectRequest);
            Console.WriteLine($"Object: {keyName} deleted from {bucketName}.");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server.
Message: '{ex.Message}' when deleting an object.");
        }
    }
}

```

Supprimez un objet dans un compartiment S3 soumis à la gestion des versions.

```

using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example creates an object in an Amazon Simple Storage Service

```

```

/// (Amazon S3) bucket and then deletes the object version that was
/// created.
/// </summary>
public class DeleteObjectVersion
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "versioned-object.txt";

        // If the AWS Region of the default user is different from the AWS
        // Region of the Amazon S3 bucket, pass the AWS Region of the
        // bucket region to the Amazon S3 client object's constructor.
        // Define it like this:
        //      RegionEndpoint bucketRegion = RegionEndpoint.USSouth1;
        IAmazonS3 client = new AmazonS3Client();

        await CreateAndDeleteObjectVersionAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// This method creates and then deletes a versioned object.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// create and delete the object.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object will be created and deleted.</param>
    /// <param name="keyName">The key name of the object to create.</param>
    public static async Task CreateAndDeleteObjectVersionAsync(IAmazonS3 client,
string bucketName, string keyName)
    {
        try
        {
            // Add a sample object.
            string versionID = await PutAnObject(client, bucketName, keyName);

            // Delete the object by specifying an object key and a version ID.
            DeleteObjectRequest request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = keyName,
                VersionId = versionID,
            };

```

```

        Console.WriteLine("Deleting an object");
        await client.DeleteObjectAsync(request);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

/// <summary>
/// This method is used to create the temporary Amazon S3 object.
/// </summary>
/// <param name="client">The initialized Amazon S3 object which will be used
/// to create the temporary Amazon S3 object.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
object
/// will be created.</param>
/// <param name="objectKey">The name of the Amazon S3 object to create.</
param>
/// <returns>The Version ID of the created object.</returns>
public static async Task<string> PutAnObject(IAmazonS3 client, string
bucketName, string objectKey)
{
    PutObjectRequest request = new PutObjectRequest()
    {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!",
    };

    PutObjectResponse response = await client.PutObjectAsync(request);
    return response.VersionId;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteObject](#) à la section Référence des AWS SDK pour .NET API.

DeleteObjects

L'exemple de code suivant montre comment utiliser `DeleteObjects`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Supprimez plusieurs objets dans un compartiment S3 non soumis à la gestion des versions.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete multiple objects from an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    /// <summary>
    /// The Main method initializes the Amazon S3 client and the name of
    /// the bucket and then passes those values to MultiObjectDeleteAsync.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "amzn-s3-demo-bucket";

        // If the Amazon S3 bucket from which you wish to delete objects is not
        // located in the same AWS Region as the default user, define the
        // AWS Region for the Amazon S3 bucket as a parameter to the client
        // constructor.
        IAmazonS3 s3Client = new AmazonS3Client();

        await MultiObjectDeleteAsync(s3Client, bucketName);
    }

    /// <summary>
    /// This method uses the passed Amazon S3 client to first create and then
    /// delete three files from the named bucket.
    /// </summary>
}
```

```

call    /// <param name="client">The initialized Amazon S3 client object used to
        /// Amazon S3 methods.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where objects
        /// will be created and then deleted.</param>
        public static async Task MultiObjectDeleteAsync(IAmazonS3 client, string
bucketName)
        {
            // Create three sample objects which we will then delete.
            var keysAndVersions = await PutObjectsAsync(client, 3, bucketName);

            // Now perform the multi-object delete, passing the key names and
            // version IDs. Since we are working with a non-versioned bucket,
            // the object keys collection includes null version IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keysAndVersions,
            };

            // You can add a specific object key to the delete request using the
            // AddKey method of the multiObjectDeleteRequest.
            try
            {
                DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionErrorStatus(e);
            }
        }

        /// <summary>
        /// Prints the list of errors raised by the call to DeleteObjectsAsync.
        /// </summary>
        /// <param name="ex">A collection of exceptions returned by the call to
        /// DeleteObjectsAsync.</param>
        public static void PrintDeletionErrorStatus(DeleteObjectsException ex)
        {
            DeleteObjectsResponse errorResponse = ex.Response;
            Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);
        }
    }

```

```

        Console.WriteLine($"Successfully deleted
{errorResponse.DeletedObjects.Count}.");
        Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");

        Console.WriteLine("Printing error data...");
        foreach (DeleteError deleteError in errorResponse.DeleteErrors)
        {
            Console.WriteLine($"Object Key:
{deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
        }
    }

    /// <summary>
    /// This method creates simple text file objects that can be used in
    /// the delete method.
    /// </summary>
    /// <param name="client">The Amazon S3 client used to call PutObjectAsync.</
param>
    /// <param name="number">The number of objects to create.</param>
    /// <param name="bucketName">The name of the bucket where the objects
    /// will be created.</param>
    /// <returns>A list of keys (object keys) and versions that the calling
    /// method will use to delete the newly created files.</returns>
    public static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3 client,
int number, string bucketName)
    {
        List<KeyVersion> keys = new List<KeyVersion>();
        for (int i = 0; i < number; i++)
        {
            string key = "ExampleObject-" + new System.Random().Next();
            PutObjectRequest request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = "This is the content body!",
            };

            PutObjectResponse response = await client.PutObjectAsync(request);

            // For non-versioned bucket operations, we only need the
            // object key.
            KeyVersion keyVersion = new KeyVersion

```



```

        {
            Key = key,
        };
        keys.Add(keyVersion);
    }

    return keys;
}
}

```

Supprimez plusieurs objets dans un compartiment S3 soumis à la gestion des versions.

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete objects in a version-enabled Amazon
/// Simple StorageService (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";

        // If the AWS Region for your Amazon S3 bucket is different from
        // the AWS Region of the default user, define the AWS Region for
        // the Amazon S3 bucket and pass it to the client constructor
        // like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        IAmazonS3 s3Client;

        s3Client = new AmazonS3Client();
        await DeleteMultipleObjectsFromVersionedBucketAsync(s3Client,
bucketName);
    }

    /// <summary>

```

```

    /// This method removes multiple versions and objects from a
    /// version-enabled Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    public static async Task
DeleteMultipleObjectsFromVersionedBucketAsync(IAmazonS3 client, string bucketName)
    {
        // Delete objects (specifying object version in the request).
        await DeleteObjectVersionsAsync(client, bucketName);

        // Delete objects (without specifying object version in the request).
        var deletedObjects = await DeleteObjectsAsync(client, bucketName);

        // Additional exercise - remove the delete markers Amazon S3 returned
from
        // the preceding response. This results in the objects reappearing
        // in the bucket (you can verify the appearance/disappearance of
        // objects in the console).
        await RemoveDeleteMarkersAsync(client, bucketName, deletedObjects);
    }

    /// <summary>
    /// Creates and then deletes non-versioned Amazon S3 objects and then
deletes
    /// them again. The method returns a list of the Amazon S3 objects deleted.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// PubObjectsAsync and NonVersionedDeleteAsync.</param>
    /// <param name="bucketName">The name of the bucket where the objects
    /// will be created and then deleted.</param>
    /// <returns>A list of DeletedObjects.</returns>
    public static async Task<List<DeletedObject>> DeleteObjectsAsync(IAmazonS3
client, string bucketName)
    {
        // Upload the sample objects.
        var keysAndVersions2 = await PutObjectsAsync(client, bucketName, 3);

        // Delete objects using only keys. Amazon S3 creates a delete marker and

```

```

        // returns its version ID in the response.
        List<DeletedObject> deletedObjects = await
NonVersionedDeleteAsync(client, bucketName, keysAndVersions2);
        return deletedObjects;
    }

    /// <summary>
    /// This method creates several temporary objects and then deletes them.
    /// </summary>
    /// <param name="client">The S3 client.</param>
    /// <param name="bucketName">Name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteObjectVersionsAsync(IAmazonS3 client, string
bucketName)
    {
        // Upload the sample objects.
        var keysAndVersions1 = await PutObjectsAsync(client, bucketName, 3);

        // Delete the specific object versions.
        await VersionedDeleteAsync(client, bucketName, keysAndVersions1);
    }

    /// <summary>
    /// Displays the list of information about deleted files to the console.
    /// </summary>
    /// <param name="e">Error information from the delete process.</param>
    private static void DisplayDeletionErrors(DeleteObjectsException e)
    {
        var errorResponse = e.Response;
        Console.WriteLine($"No. of objects successfully deleted =
{errorResponse.DeletedObjects.Count}");
        Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");
        Console.WriteLine("Printing error data...");
        foreach (var deleteError in errorResponse.DeleteErrors)
        {
            Console.WriteLine($"Object Key:
{deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
        }
    }

    /// <summary>
    /// Delete multiple objects from a version-enabled bucket.
    /// </summary>

```

```

    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="keys">A list of key names for the objects to delete.</
param>
    private static async Task VersionedDeleteAsync(IAmazonS3 client, string
bucketName, List<KeyVersion> keys)
    {
        var multiObjectDeleteRequest = new DeleteObjectsRequest
        {
            BucketName = bucketName,
            Objects = keys, // This includes the object keys and specific
version IDs.
        };

        try
        {
            Console.WriteLine("Executing VersionedDelete...");
            DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine($"Successfully deleted all the
{response.DeletedObjects.Count} items");
        }
        catch (DeleteObjectsException ex)
        {
            DisplayDeletionErrors(ex);
        }
    }

    /// <summary>
    /// Deletes multiple objects from a non-versioned Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
    /// RemoveDeleteMarkersAsync.</param>
    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="keys">A list of key names for the objects to delete.</
param>
    /// <returns>A list of the deleted objects.</returns>

```

```

        private static async Task<List<DeletedObject>>
NonVersionedDeleteAsync(IAmazonS3 client, string bucketName, List<KeyVersion> keys)
        {
            // Create a request that includes only the object key names.
            DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest();
            multiObjectDeleteRequest.BucketName = bucketName;

            foreach (var key in keys)
            {
                multiObjectDeleteRequest.AddKey(key.Key);
            }

            // Execute DeleteObjectsAsync.
            // The DeleteObjectsAsync method adds a delete marker for each
            // object deleted. You can verify that the objects were removed
            // using the Amazon S3 console.
            DeleteObjectsResponse response;
            try
            {
                Console.WriteLine("Executing NonVersionedDelete...");
                response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
                throw; // Some deletions failed. Investigate before continuing.
            }

            // This response contains the DeletedObjects list which we use to delete
            the delete markers.
            return response.DeletedObjects;
        }

        /// <summary>
        /// Deletes the markers left after deleting the temporary objects.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>

```

```

    /// <param name="bucketName">The name of the bucket from which to delete
    /// objects.</param>
    /// <param name="deletedObjects">A list of the objects that were deleted.</
param>
    private static async Task RemoveDeleteMarkersAsync(IAmazonS3 client, string
bucketName, List<DeletedObject> deletedObjects)
    {
        var keyVersionList = new List<KeyVersion>();

        foreach (var deletedObject in deletedObjects)
        {
            KeyVersion keyVersion = new KeyVersion
            {
                Key = deletedObject.Key,
                VersionId = deletedObject.DeleteMarkerVersionId,
            };
            keyVersionList.Add(keyVersion);
        }

        // Create another request to delete the delete markers.
        var multiObjectDeleteRequest = new DeleteObjectsRequest
        {
            BucketName = bucketName,
            Objects = keyVersionList,
        };

        // Now, delete the delete marker to bring your objects back to the
bucket.
        try
        {
            Console.WriteLine("Removing the delete markers .....");
            var deleteObjectResponse = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
            Console.WriteLine($"Successfully deleted the
{deleteObjectResponse.DeletedObjects.Count} delete markers");
        }
        catch (DeleteObjectsException ex)
        {
            DisplayDeletionErrors(ex);
        }
    }

    /// <summary>

```

```

    /// Create temporary Amazon S3 objects to show how object deletion works in
an
    /// Amazon S3 bucket with versioning enabled.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// PutObjectAsync to create temporary objects for the example.</param>
    /// <param name="bucketName">A string representing the name of the S3
    /// bucket where we will create the temporary objects.</param>
    /// <param name="number">The number of temporary objects to create.</param>
    /// <returns>A list of the KeyVersion objects.</returns>
    private static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3
client, string bucketName, int number)
    {
        var keys = new List<KeyVersion>();

        for (var i = 0; i < number; i++)
        {
            string key = "ObjectToDelete-" + new System.Random().Next();
            PutObjectRequest request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = "This is the content body!",
            };

            var response = await client.PutObjectAsync(request);
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
                VersionId = response.VersionId,
            };

            keys.Add(keyVersion);
        }

        return keys;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DeleteObjects](#) à la section Référence des AWS SDK pour .NET API.

GetBucketAcl

L'exemple de code suivant montre comment utiliser `GetBucketAcl`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the access control list (ACL) for the new bucket.
/// </summary>
/// <param name="client">The initialized client object used to get the
/// access control list (ACL) of the bucket.</param>
/// <param name="newBucketName">The name of the newly created bucket.</
param>
/// <returns>An S3AccessControlList.</returns>
public static async Task<S3AccessControlList> GetACLForBucketAsync(IAmazonS3
client, string newBucketName)
{
    // Retrieve bucket ACL to show that the ACL was properly applied to
    // the new bucket.
    GetACLResponse getACLResponse = await client.GetACLAsync(new
GetACLRequest
    {
        BucketName = newBucketName,
    });

    return getACLResponse.AccessControlList;
}
```


- Pour plus de détails sur l'API, reportez-vous [GetBucketAcl](#) à la section Référence des AWS SDK pour .NET API.

GetBucketCors

L'exemple de code suivant montre comment utiliser `GetBucketCors`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Retrieve the CORS configuration applied to the Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used
/// to retrieve the CORS configuration.</param>
/// <returns>The created CORS configuration object.</returns>
private static async Task<CORSConfiguration>
RetrieveCORSConfigurationAsync(AmazonS3Client client)
{
    GetCORSConfigurationRequest request = new GetCORSConfigurationRequest()
    {
        BucketName = BucketName,
    };
    var response = await client.GetCORSConfigurationAsync(request);
    var configuration = response.Configuration;
    PrintCORSRules(configuration);
    return configuration;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetBucketCors](#) à la section Référence des AWS SDK pour .NET API.

GetBucketEncryption

L'exemple de code suivant montre comment utiliser `GetBucketEncryption`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get and print the encryption settings of a bucket.
/// </summary>
/// <param name="bucketName">Name of the bucket.</param>
/// <returns>Async task.</returns>
public static async Task GetEncryptionSettings(string bucketName)
{
    // Check and print the bucket encryption settings.
    Console.WriteLine($"Getting encryption settings for bucket {bucketName}.");

    try
    {
        var settings =
            await _s3Client.GetBucketEncryptionAsync(
                new GetBucketEncryptionRequest() { BucketName = bucketName });

        foreach (var encryptionSettings in
            settings?.ServerSideEncryptionConfiguration?.ServerSideEncryptionRules!)
        {
            Console.WriteLine(
                $"{Environment.NewLine}\tAlgorithm:
{encryptionSettings.ServerSideEncryptionByDefault.ServerSideEncryptionAlgorithm}");
            Console.WriteLine(
                $"{Environment.NewLine}\tKey:
{encryptionSettings.ServerSideEncryptionByDefault.ServerSideEncryptionKeyManagementServiceK
            }
        }
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine(ex.ErrorCode == "InvalidBucketName"
            ? $"Bucket {bucketName} was not found."
            );
    }
}
```

```

        : $"Unable to get bucket encryption for bucket {bucketName},
{ex.Message}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [GetBucketEncryption](#) à la section Référence des AWS SDK pour .NET API.

GetBucketLifecycleConfiguration

L'exemple de code suivant montre comment utiliser `GetBucketLifecycleConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Returns a configuration object for the supplied bucket name.
    /// </summary>
    /// <param name="client">The S3 client object used to call
    /// the GetLifecycleConfigurationAsync method.</param>
    /// <param name="bucketName">The name of the S3 bucket for which a
    /// configuration will be created.</param>
    /// <returns>Returns a new LifecycleConfiguration object.</returns>
    public static async Task<LifecycleConfiguration>
RetrieveLifecycleConfigAsync(IAmazonS3 client, string bucketName)
    {
        var request = new GetLifecycleConfigurationRequest()
        {
            BucketName = bucketName,
        };
        var response = await client.GetLifecycleConfigurationAsync(request);
        var configuration = response.Configuration;
        return configuration;
    }

```

- Pour plus de détails sur l'API, reportez-vous [GetBucketLifecycleConfiguration](#) à la section Référence des AWS SDK pour .NET API.

GetBucketWebsite

L'exemple de code suivant montre comment utiliser `GetBucketWebsite`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
// Get the website configuration.
GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
{
    BucketName = bucketName,
};
GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
Console.WriteLine($"Index document:
{getResponse.WebsiteConfiguration.IndexDocumentSuffix}");
Console.WriteLine($"Error document:
{getResponse.WebsiteConfiguration.ErrorDocument}");
```

- Pour plus de détails sur l'API, reportez-vous [GetBucketWebsite](#) à la section Référence des AWS SDK pour .NET API.

GetObject

L'exemple de code suivant montre comment utiliser `GetObject`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Obtenez un objet à l'aide d'une demande conditionnelle.

```
/// <summary>
/// Retrieves an object from Amazon S3 with a conditional request.
/// </summary>
/// <param name="objectKey">The key of the object to retrieve.</param>
/// <param name="sourceBucket">The source bucket of the object.</param>
/// <param name="conditionType">The type of condition: 'IfMatch', 'IfNoneMatch',
'IfModifiedSince', 'IfUnmodifiedSince'.</param>
/// <param name="conditionDateValue">The value to use for the condition for
dates.</param>
/// <param name="etagConditionalValue">The value to use for the condition for
etags.</param>
/// <returns>True if the conditional read is successful, False otherwise.</
returns>
public async Task<bool> GetObjectConditional(string objectKey, string
sourceBucket,
    S3ConditionType conditionType, DateTime? conditionDateValue = null, string?
etagConditionalValue = null)
{
    try
    {
        var getObjectRequest = new GetObjectRequest
        {
            BucketName = sourceBucket,
            Key = objectKey
        };

        switch (conditionType)
        {
            case S3ConditionType.IfMatch:
                getObjectRequest.EtagToMatch = etagConditionalValue;
                break;
            case S3ConditionType.IfNoneMatch:
                getObjectRequest.EtagToNotMatch = etagConditionalValue;
```

```

        break;
        case S3ConditionType.IfModifiedSince:
            getObjectRequest.ModifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
            break;
        case S3ConditionType.IfUnmodifiedSince:
            getObjectRequest.UnmodifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
            break;
        default:
            throw new ArgumentOutOfRangeException(nameof(conditionType),
conditionType, null);
    }

    var response = await _amazonS3.GetObjectAsync(getObjectRequest);
    var sampleBytes = new byte[20];
    await response.ResponseStream.ReadAsync(sampleBytes, 0, 20);
    _logger.LogInformation($"Conditional read successful. Here are the first
20 bytes of the object:\n{System.Text.Encoding.UTF8.GetString(sampleBytes)}");
    return true;
}
catch (AmazonS3Exception e)
{
    if (e.ErrorCode == "PreconditionFailed")
    {
        _logger.LogError("Conditional read failed: Precondition failed");
    }
    else if (e.ErrorCode == "NotModified")
    {
        _logger.LogError("Conditional read failed: Object not modified");
    }
    else
    {
        _logger.LogError($"Unexpected error: {e.ErrorCode}");
        throw;
    }
    return false;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [GetObject](#) à la section Référence des AWS SDK pour .NET API.

GetObjectLegalHold

L'exemple de code suivant montre comment utiliser `GetObjectLegalHold`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the legal hold details for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object legal hold details.</returns>
public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectLegalHoldAsync(request);
        Console.WriteLine($"Object legal hold for {objectKey} in {bucketName}:

" +
            $"{response.LegalHold.Status}");
        return response.LegalHold;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Unable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [GetObjectLegalHold](#) à la section Référence des AWS SDK pour .NET API.

GetObjectLockConfiguration

L'exemple de code suivant montre comment utiliser `GetObjectLockConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the object lock configuration details for an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket to get details.</param>
/// <returns>The bucket's object lock configuration details.</returns>
public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
{
    try
    {
        var request = new GetObjectLockConfigurationRequest()
        {
            BucketName = bucketName
        };

        var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
        Console.WriteLine($"{bucketName} object lock config for {bucketName} in
{bucketName}: " +
                        $"{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                        $"{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

        return response.ObjectLockConfiguration;
    }
    catch (AmazonS3Exception ex)
    {

```



```
        Console.WriteLine($"\\tUnable to fetch object lock config:
'{ex.Message}'");
        return new ObjectLockConfiguration();
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [GetObjectLockConfiguration](#) à la section Référence des AWS SDK pour .NET API.

GetObjectRetention

L'exemple de code suivant montre comment utiliser `GetObjectRetention`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the retention period for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object retention details.</returns>
public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectRetentionAsync(request);
    }
}
```

```

        Console.WriteLine($"\\tObject retention for {objectKey} in {bucketName}:
" +
        $"\\n\\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
        return response.Retention;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to fetch object lock retention:
'{ex.Message}'");
        return new ObjectLockRetention();
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [GetObjectRetention](#) à la section Référence des AWS SDK pour .NET API.

ListBuckets

L'exemple de code suivant montre comment utiliser `ListBuckets`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

namespace ListBucketsExample
{
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example uses the AWS SDK for .NET to list the Amazon Simple Storage
    /// Service (Amazon S3) buckets belonging to the default account.

```

```

/// </summary>
public class ListBuckets
{
    private static IAmazonS3 _s3Client;

    /// <summary>
    /// Get a list of the buckets owned by the default user.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client object.</param>
    /// <returns>The response from the ListingBuckets call that contains a
    /// list of the buckets owned by the default user.</returns>
    public static async Task<ListBucketsResponse> GetBuckets(IAmazonS3 client)
    {
        return await client.ListBucketsAsync();
    }

    /// <summary>
    /// This method lists the name and creation date for the buckets in
    /// the passed List of S3 buckets.
    /// </summary>
    /// <param name="bucketList">A List of S3 bucket objects.</param>
    public static void DisplayBucketList(List<S3Bucket> bucketList)
    {
        bucketList
            .ForEach(b => Console.WriteLine($"Bucket name: {b.BucketName},
created on: {b.CreationDate}"));
    }

    public static async Task Main()
    {
        // The client uses the AWS Region of the default user.
        // If the Region where the buckets were created is different,
        // pass the Region to the client constructor. For example:
        // _s3Client = new AmazonS3Client(RegionEndpoint.USEast1);
        _s3Client = new AmazonS3Client();
        var response = await GetBuckets(_s3Client);
        DisplayBucketList(response.Buckets);
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListBuckets](#) à la section Référence des AWS SDK pour .NET API.

ListObjectVersions

L'exemple de code suivant montre comment utiliser `ListObjectVersions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example lists the versions of the objects in a version enabled
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class ListObjectVersions
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";

        // If the AWS Region where your bucket is defined is different from
        // the AWS Region where the Amazon S3 bucket is defined, pass the
constant
        // for the AWS Region to the client constructor like this:
        //     var client = new AmazonS3Client(RegionEndpoint.USSouth1);
        IAmazonS3 client = new AmazonS3Client();
        await GetObjectListWithAllVersionsAsync(client, bucketName);
    }

    /// <summary>
    /// This method lists all versions of the objects within an Amazon S3
    /// version enabled bucket.
```

```

    /// </summary>
    /// <param name="client">The initialized client object used to call
    /// ListVersionsAsync.</param>
    /// <param name="bucketName">The name of the version enabled Amazon S3
bucket
    /// for which you want to list the versions of the contained objects.</
param>
    public static async Task GetObjectListWithAllVersionsAsync(IAmazonS3 client,
string bucketName)
    {
        try
        {
            // When you instantiate the ListVersionRequest, you can
            // optionally specify a key name prefix in the request
            // if you want a list of object versions of a specific object.

            // For this example we set a small limit in MaxKeys to return
            // a small list of versions.
            ListVersionsRequest request = new ListVersionsRequest()
            {
                BucketName = bucketName,
                MaxKeys = 2,
            };

            do
            {
                ListVersionsResponse response = await
client.ListVersionsAsync(request);

                // Process response.
                foreach (S3ObjectVersion entry in response.Versions)
                {
                    Console.WriteLine($"key: {entry.Key} size: {entry.Size}");
                }

                // If response is truncated, set the marker to get the next
                // set of keys.
                if (response.IsTruncated)
                {
                    request.KeyMarker = response.NextKeyMarker;
                    request.VersionIdMarker = response.NextVersionIdMarker;
                }
            }
            else
            {

```

```

        request = null;
    }
    }
    while (request != null);
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error: '{ex.Message}'");
}
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ListObjectVersions](#) à la section Référence des AWS SDK pour .NET API.

ListObjectsV2

L'exemple de code suivant montre comment utiliser `ListObjectsV2`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Listez les objets avec un paginateur.

```

using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// The following example lists objects in an Amazon Simple Storage
/// Service (Amazon S3) bucket.
/// </summary>
public class ListObjectsPaginator

```

```

{
    private const string BucketName = "amzn-s3-demo-bucket";

    public static async Task Main()
    {
        IAmazonS3 s3Client = new AmazonS3Client();

        Console.WriteLine($"Listing the objects contained in {BucketName}:\n");
        await ListingObjectsAsync(s3Client, BucketName);
    }

    /// <summary>
    /// This method uses a paginator to retrieve the list of objects in an
    /// an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">An Amazon S3 client object.</param>
    /// <param name="bucketName">The name of the S3 bucket whose objects
    /// you want to list.</param>
    public static async Task ListingObjectsAsync(IAmazonS3 client, string
bucketName)
    {
        var listObjectsV2Paginator = client.Paginators.ListObjectsV2(new
ListObjectsV2Request
        {
            BucketName = bucketName,
        });

        await foreach (var response in listObjectsV2Paginator.Responses)
        {
            Console.WriteLine($"HttpStatusCode: {response.HttpStatusCode}");
            Console.WriteLine($"Number of Keys: {response.KeyCount}");
            foreach (var entry in response.S3Objects)
            {
                Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
            }
        }
    }
}

```

- Pour plus de détails sur l'API, voir [ListObjectsV2](#) dans le manuel de référence des AWS SDK pour .NET API.

PutBucketAccelerateConfiguration

L'exemple de code suivant montre comment utiliser `PutBucketAccelerateConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Amazon Simple Storage Service (Amazon S3) Transfer Acceleration is a
/// bucket-level feature that enables you to perform faster data transfers
/// to Amazon S3. This example shows how to configure Transfer
/// Acceleration.
/// </summary>
public class TransferAcceleration
{
    /// <summary>
    /// The main method initializes the client object and sets the
    /// Amazon Simple Storage Service (Amazon S3) bucket name before
    /// calling EnableAccelerationAsync.
    /// </summary>
    public static async Task Main()
    {
        var s3Client = new AmazonS3Client();
        const string bucketName = "amzn-s3-demo-bucket";

        await EnableAccelerationAsync(s3Client, bucketName);
    }

    /// <summary>
    /// This method sets the configuration to enable transfer acceleration
    /// for the bucket referred to in the bucketName parameter.
    /// </summary>
    /// <param name="client">An Amazon S3 client used to enable the
```



```

    /// acceleration on an Amazon S3 bucket.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket for which the
    /// method will be enabling acceleration.</param>
    private static async Task EnableAccelerationAsync(AmazonS3Client client,
string bucketName)
    {
        try
        {
            var putRequest = new PutBucketAccelerateConfigurationRequest
            {
                BucketName = bucketName,
                AccelerateConfiguration = new AccelerateConfiguration
                {
                    Status = BucketAccelerateStatus.Enabled,
                },
            };
            await client.PutBucketAccelerateConfigurationAsync(putRequest);

            var getRequest = new GetBucketAccelerateConfigurationRequest
            {
                BucketName = bucketName,
            };
            var response = await
client.GetBucketAccelerateConfigurationAsync(getRequest);

            Console.WriteLine($"Acceleration state = '{response.Status}' ");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error occurred. Message: '{ex.Message}' when
setting transfer acceleration");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [PutBucketAccelerateConfiguration](#) à la section Référence des AWS SDK pour .NET API.

PutBucketAc1

L'exemple de code suivant montre comment utiliser PutBucketAc1.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Creates an Amazon S3 bucket with an ACL to control access to the
    /// bucket and the objects stored in it.
    /// </summary>
    /// <param name="client">The initialized client object used to create
    /// an Amazon S3 bucket, with an ACL applied to the bucket.
    /// </param>
    /// <param name="region">The AWS Region where the bucket will be created.</
param>
    /// <param name="newBucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value indicating success or failure.</returns>
    public static async Task<bool> CreateBucketUseCannedACLAsync(IAmazonS3
client, S3Region region, string newBucketName)
    {
        try
        {
            // Create a new Amazon S3 bucket with Canned ACL.
            var putBucketRequest = new PutBucketRequest()
            {
                BucketName = newBucketName,
                BucketRegion = region,
                CannedACL = S3CannedACL.LogDeliveryWrite,
            };

            PutBucketResponse putBucketResponse = await
client.PutBucketAsync(putBucketRequest);

            return putBucketResponse.HttpStatusCode ==
System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Amazon S3 error: {ex.Message}");
        }
    }

```

```
    }  
  
    return false;  
}
```

- Pour plus de détails sur l'API, reportez-vous [PutBucketAcl](#) à la section Référence des AWS SDK pour .NET API.

PutBucketCors

L'exemple de code suivant montre comment utiliser `PutBucketCors`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Add CORS configuration to the Amazon S3 bucket.  
/// </summary>  
/// <param name="client">The initialized Amazon S3 client object used  
/// to apply the CORS configuration to an Amazon S3 bucket.</param>  
/// <param name="configuration">The CORS configuration to apply.</param>  
private static async Task PutCORSConfigurationAsync(AmazonS3Client client,  
CORSConfiguration configuration)  
{  
    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest()  
    {  
        BucketName = BucketName,  
        Configuration = configuration,  
    };  
  
    _ = await client.PutCORSConfigurationAsync(request);  
}
```

- Pour plus de détails sur l'API, reportez-vous [PutBucketCors](#) à la section Référence des AWS SDK pour .NET API.

PutBucketEncryption

L'exemple de code suivant montre comment utiliser `PutBucketEncryption`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Set the bucket server side encryption to use AWSKMS with a customer-managed
key id.
/// </summary>
/// <param name="bucketName">Name of the bucket.</param>
/// <param name="kmsKeyId">The Id of the KMS Key.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> SetBucketServerSideEncryption(string bucketName,
string kmsKeyId)
{
    var serverSideEncryptionByDefault = new ServerSideEncryptionConfiguration
    {
        ServerSideEncryptionRules = new List<ServerSideEncryptionRule>
        {
            new ServerSideEncryptionRule
            {
                ServerSideEncryptionByDefault = new
ServerSideEncryptionByDefault
                {
                    ServerSideEncryptionAlgorithm =
ServerSideEncryptionMethod.AWSKMS,
                    ServerSideEncryptionKeyManagementServiceKeyId = kmsKeyId
                }
            }
        }
    }
}
```

```
    }  
    };  
    try  
    {  
        var encryptionResponse = await _s3Client.PutBucketEncryptionAsync(new  
PutBucketEncryptionRequest  
        {  
            BucketName = bucketName,  
            ServerSideEncryptionConfiguration = serverSideEncryptionByDefault,  
        });  
  
        return encryptionResponse.HttpStatusCode == HttpStatusCode.OK;  
    }  
    catch (AmazonS3Exception ex)  
    {  
        Console.WriteLine(ex.ErrorCode == "AccessDenied"  
            ? $"This account does not have permission to set encryption on  
{bucketName}, please try again."  
            : $"Unable to set bucket encryption for bucket {bucketName},  
{ex.Message}");  
    }  
    return false;  
}
```

- Pour plus de détails sur l'API, reportez-vous [PutBucketEncryption](#) à la section Référence des AWS SDK pour .NET API.

PutBucketLifecycleConfiguration

L'exemple de code suivant montre comment utiliser `PutBucketLifecycleConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Adds lifecycle configuration information to the S3 bucket named in
    /// the bucketName parameter.
    /// </summary>
    /// <param name="client">The S3 client used to call the
    /// PutLifecycleConfigurationAsync method.</param>
    /// <param name="bucketName">A string representing the S3 bucket to
    /// which configuration information will be added.</param>
    /// <param name="configuration">A LifecycleConfiguration object that
    /// will be applied to the S3 bucket.</param>
    public static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
string bucketName, LifecycleConfiguration configuration)
    {
        var request = new PutLifecycleConfigurationRequest()
        {
            BucketName = bucketName,
            Configuration = configuration,
        };
        var response = await client.PutLifecycleConfigurationAsync(request);
    }

```

- Pour plus de détails sur l'API, reportez-vous [PutBucketLifecycleConfiguration](#) à la section Référence des AWS SDK pour .NET API.

PutBucketLogging

L'exemple de code suivant montre comment utiliser PutBucketLogging.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.IO;
using System.Threading.Tasks;

```

```

using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

/// <summary>
/// This example shows how to enable logging on an Amazon Simple Storage
/// Service (Amazon S3) bucket. You need to have two Amazon S3 buckets for
/// this example. The first is the bucket for which you wish to enable
/// logging, and the second is the location where you want to store the
/// logs.
/// </summary>
public class ServerAccessLogging
{
    private static IConfiguration _configuration = null!;

    public static async Task Main()
    {
        LoadConfig();

        string bucketName = _configuration["BucketName"];
        string logBucketName = _configuration["LogBucketName"];
        string logObjectKeyPrefix = _configuration["LogObjectKeyPrefix"];
        string accountId = _configuration["AccountId"];

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        try
        {
            // Update bucket policy for target bucket to allow delivery of logs
            to it.
            await SetBucketPolicyToAllowLogDelivery(
                client,
                bucketName,
                logBucketName,
                logObjectKeyPrefix,
                accountId);

            // Enable logging on the source bucket.
            await EnableLoggingAsync(
                client,

```

```

        bucketName,
        logBucketName,
        logObjectKeyPrefix);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine($"Error: {e.Message}");
    }
}

/// <summary>
/// This method grants appropriate permissions for logging to the
/// Amazon S3 bucket where the logs will be stored.
/// </summary>
/// <param name="client">The initialized Amazon S3 client which will be used
/// to apply the bucket policy.</param>
/// <param name="sourceBucketName">The name of the source bucket.</param>
/// <param name="logBucketName">The name of the bucket where logging
/// information will be stored.</param>
/// <param name="logPrefix">The logging prefix where the logs should be
delivered.</param>
/// <param name="accountId">The account id of the account where the source
bucket exists.</param>
/// <returns>Async task.</returns>
public static async Task SetBucketPolicyToAllowLogDelivery(
    IAmazonS3 client,
    string sourceBucketName,
    string logBucketName,
    string logPrefix,
    string accountId)
{
    var resourceArn = @""arn:aws:s3:::" + logBucketName + "/" + logPrefix +
@"*""";

    var newPolicy = @"{
        ""Statement"": [{
            ""Sid"": ""S3ServerAccessLogsPolicy"",
            ""Effect"": ""Allow"",
            ""Principal"": { ""Service"":
""logging.s3.amazonaws.com"" },
            ""Action"": [""s3:PutObject""],
            ""Resource"": ["" + resourceArn + @""],
            ""Condition"": {

```



```

        ""ArnLike"": { ""aws:SourceArn"": ""arn:aws:s3:::" +
sourceBucketName + @"""" },
        ""StringEquals"": { ""aws:SourceAccount"": """" +
accountId + @"""" }
    }
}]]
}";

    Console.WriteLine($"The policy to apply to bucket {logBucketName} to
enable logging:");
    Console.WriteLine(newPolicy);

    PutBucketPolicyRequest putRequest = new PutBucketPolicyRequest
    {
        BucketName = logBucketName,
        Policy = newPolicy,
    };
    await client.PutBucketPolicyAsync(putRequest);
    Console.WriteLine("Policy applied.");
}

/// <summary>
/// This method enables logging for an Amazon S3 bucket. Logs will be stored
/// in the bucket you selected for logging. Selected prefix
/// will be prepended to each log object.
/// </summary>
/// <param name="client">The initialized Amazon S3 client which will be used
/// to configure and apply logging to the selected Amazon S3 bucket.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket for which you
/// wish to enable logging.</param>
/// <param name="logBucketName">The name of the Amazon S3 bucket where
logging
/// information will be stored.</param>
/// <param name="logObjectKeyPrefix">The prefix to prepend to each
/// object key.</param>
/// <returns>Async task.</returns>
public static async Task EnableLoggingAsync(
    IAmazonS3 client,
    string bucketName,
    string logBucketName,
    string logObjectKeyPrefix)
{
    Console.WriteLine($"Enabling logging for bucket {bucketName}.");
    var loggingConfig = new S3BucketLoggingConfig
    {

```

```

        TargetBucketName = logBucketName,
        TargetPrefix = logObjectKeyPrefix,
    };

    var putBucketLoggingRequest = new PutBucketLoggingRequest
    {
        BucketName = bucketName,
        LoggingConfig = loggingConfig,
    };
    await client.PutBucketLoggingAsync(putBucketLoggingRequest);
    Console.WriteLine($"Logging enabled.");
}

/// <summary>
/// Loads configuration from settings files.
/// </summary>
public static void LoadConfig()
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json", true) // Optionally, load local
settings.
        .Build();
}
}

```

- Pour plus de détails sur l'API, reportez-vous [PutBucketLogging](#) à la section Référence des AWS SDK pour .NET API.

PutBucketNotificationConfiguration

L'exemple de code suivant montre comment utiliser `PutBucketNotificationConfiguration`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to enable notifications for an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class EnableNotifications
{
    public static async Task Main()
    {
        const string bucketName = "amzn-s3-demo-bucket1";
        const string snsTopic = "arn:aws:sns:us-east-2:0123456789ab:bucket-
notify";
        const string sqsQueue = "arn:aws:sqs:us-
east-2:0123456789ab:Example_Queue";

        IAmazonS3 client = new AmazonS3Client(Amazon.RegionEndpoint.USEast2);
        await EnableNotificationAsync(client, bucketName, snsTopic, sqsQueue);
    }

    /// <summary>
    /// This method makes the call to the PutBucketNotificationAsync method.
    /// </summary>
    /// <param name="client">An initialized Amazon S3 client used to call
    /// the PutBucketNotificationAsync method.</param>
    /// <param name="bucketName">The name of the bucket for which
    /// notifications will be turned on.</param>
    /// <param name="snsTopic">The ARN for the Amazon Simple Notification
    /// Service (Amazon SNS) topic associated with the S3 bucket.</param>
    /// <param name="sqsQueue">The ARN of the Amazon Simple Queue Service
```

```

    /// (Amazon SQS) queue to which notifications will be pushed.</param>
    public static async Task EnableNotificationAsync(
        IAmazonS3 client,
        string bucketName,
        string snsTopic,
        string sqsQueue)
    {
        try
        {
            // The bucket for which we are setting up notifications.
            var request = new PutBucketNotificationRequest()
            {
                BucketName = bucketName,
            };

            // Defines the topic to use when sending a notification.
            var topicConfig = new TopicConfiguration()
            {
                Events = new List<EventType> { EventType.ObjectCreatedCopy },
                Topic = snsTopic,
            };
            request.TopicConfigurations = new List<TopicConfiguration>
            {
                topicConfig,
            };
            request.QueueConfigurations = new List<QueueConfiguration>
            {
                new QueueConfiguration()
                {
                    Events = new List<EventType> { EventType.ObjectCreatedPut },
                    Queue = sqsQueue,
                },
            };

            // Now apply the notification settings to the bucket.
            PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [PutBucketNotificationConfiguration](#) à la section Référence des AWS SDK pour .NET API.

PutBucketWebsite

L'exemple de code suivant montre comment utiliser `PutBucketWebsite`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
// Put the website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
{
    BucketName = bucketName,
    WebsiteConfiguration = new WebsiteConfiguration()
    {
        IndexDocumentSuffix = indexDocumentSuffix,
        ErrorDocument = errorDocument,
    },
};
PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- Pour plus de détails sur l'API, reportez-vous [PutBucketWebsite](#) à la section Référence des AWS SDK pour .NET API.

PutObject

L'exemple de code suivant montre comment utiliser `PutObject`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Chargez un objet avec un chiffrement côté serveur.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to upload an object to an Amazon Simple Storage
/// Service (Amazon S3) bucket with server-side encryption enabled.
/// </summary>
public class ServerSideEncryption
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "samplefile.txt";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();

        await WritingAnObjectAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// Upload a sample object include a setting for encryption.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// to upload a file and apply server-side encryption.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// encrypted object will reside.</param>
    /// <param name="keyName">The name for the object that you want to
```

```

    /// create in the supplied bucket.</param>
    public static async Task WritingAnObjectAsync(IAmazonS3 client, string
bucketName, string keyName)
    {
        try
        {
            var putRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
                ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256,
            };

            var putResponse = await client.PutObjectAsync(putRequest);

            // Determine the encryption state of an object.
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest
            {
                BucketName = bucketName,
                Key = keyName,
            };
            GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
            ServerSideEncryptionMethod objectEncryption =
response.ServerSideEncryptionMethod;

            Console.WriteLine($"Encryption method used: {0}",
objectEncryption.ToString());
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error: '{ex.Message}' when writing an object");
        }
    }
}

```

Placez un objet à l'aide d'une demande conditionnelle.

```
/// <summary>
```

```

    /// Uploads an object to Amazon S3 with a conditional request. Prevents
    /// overwrite using an IfNoneMatch condition for the object key.
    /// </summary>
    /// <param name="objectKey">The key of the object to upload.</param>
    /// <param name="bucket">The source bucket of the object.</param>
    /// <param name="content">The content to upload as a string.</param>
    /// <returns>The ETag if the conditional write is successful, empty otherwise.</
returns>
    public async Task<string> PutObjectConditional(string objectKey, string bucket,
string content)
    {
        try
        {
            var putObjectRequest = new PutObjectRequest
            {
                BucketName = bucket,
                Key = objectKey,
                ContentBody = content,
                IfNoneMatch = "*"
            };

            var putResult = await _amazonS3.PutObjectAsync(putObjectRequest);
            _logger.LogInformation($"Conditional write successful for key
{objectKey} in bucket {bucket}.");
            return putResult.ETag;
        }
        catch (AmazonS3Exception e)
        {
            if (e.ErrorCode == "PreconditionFailed")
            {
                _logger.LogError("Conditional write failed: Precondition failed");
            }
            else
            {
                _logger.LogError($"Unexpected error: {e.ErrorCode}");
                throw;
            }
            return string.Empty;
        }
    }
}

```


- Pour plus de détails sur l'API, reportez-vous [PutObject](#) à la section Référence des AWS SDK pour .NET API.

PutObjectLegalHold

L'exemple de code suivant montre comment utiliser `PutObjectLegalHold`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Set or modify a legal hold on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="holdStatus">The On or Off status for the legal hold.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectLegalHold(string bucketName,
    string objectKey, ObjectLockLegalHoldStatus holdStatus)
{
    try
    {
        var request = new PutObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            LegalHold = new ObjectLockLegalHold()
            {
                Status = holdStatus
            }
        };

        var response = await _amazonS3.PutObjectLegalHoldAsync(request);
        Console.WriteLine($"{bucketName}\tModified legal hold for {objectKey} in {bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

```

        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\\tError modifying legal hold: '{ex.Message}'");
            return false;
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [PutObjectLegalHold](#) à la section Référence des AWS SDK pour .NET API.

PutObjectLockConfiguration

L'exemple de code suivant montre comment utiliser `PutObjectLockConfiguration`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Définissez la configuration du verrouillage d'objet d'un compartiment.

```

/// <summary>
/// Enable object lock on an existing bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket to modify.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableObjectLockOnBucket(string bucketName)
{
    try
    {
        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,

```

```

        Status = VersionStatus.Enabled
    }
});

var request = new PutObjectLockConfigurationRequest()
{
    BucketName = bucketName,
    ObjectLockConfiguration = new ObjectLockConfiguration()
    {
        ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
    },
};

var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
Console.WriteLine($"{bucketName}\tAdded an object lock policy to bucket
{bucketName}.");
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
    return false;
}
}

```

Définissez la période de rétention par défaut d'un compartiment.

```

/// <summary>
/// Set or modify a retention period on an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket to modify.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date for retention until.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
{
    var enabledString = enableObjectLock ? "Enabled" : "Disabled";
    var timeDifference = retainUntilDate.Subtract(DateTime.Now);
    try
    {
        // First, enable Versioning on the bucket.
    }
}

```

```

        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
    {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig()
        {
            EnableMfaDelete = false,
            Status = VersionStatus.Enabled
        }
    });

var request = new PutObjectLockConfigurationRequest()
{
    BucketName = bucketName,
    ObjectLockConfiguration = new ObjectLockConfiguration()
    {
        ObjectLockEnabled = new ObjectLockEnabled(enabledString),
        Rule = new ObjectLockRule()
        {
            DefaultRetention = new DefaultRetention()
            {
                Mode = retention,
                Days = timeDifference.Days // Can be specified in days
or years but not both.
            }
        }
    }
};

var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
Console.WriteLine($"\\tAdded a default retention to bucket
{bucketName}.");
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"\\tError modifying object lock: '{ex.Message}'");
    return false;
}
}

```

- Pour plus de détails sur l'API, reportez-vous [PutObjectLockConfiguration](#) à la section Référence des AWS SDK pour .NET API.

PutObjectRetention

L'exemple de code suivant montre comment utiliser `PutObjectRetention`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Set or modify a retention period on an object in an S3 bucket.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The key of the object.</param>
/// <param name="retention">The retention mode.</param>
/// <param name="retainUntilDate">The date retention expires.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
    string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
{
    try
    {
        var request = new PutObjectRetentionRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            Retention = new ObjectLockRetention()
            {
                Mode = retention,
                RetainUntilDate = retainUntilDate
            }
        };

        var response = await _amazonS3.PutObjectRetentionAsync(request);
```

```

        Console.WriteLine($"\\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying retention period:
'{ex.Message}'");
        return false;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [PutObjectRetention](#) à la section Référence des AWS SDK pour .NET API.

RestoreObject

L'exemple de code suivant montre comment utiliser `RestoreObject`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to restore an archived object in an Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class RestoreArchivedObject
{
    public static void Main()

```

```

    {
        string bucketName = "amzn-s3-demo-bucket";
        string objectKey = "archived-object.txt";

        // Specify your bucket region (an example region is shown).
        RegionEndpoint bucketRegion = RegionEndpoint.USSouth1;

        IAmazonS3 client = new AmazonS3Client(bucketRegion);
        RestoreObjectAsync(client, bucketName, objectKey).Wait();
    }

    /// <summary>
    /// This method restores an archived object from an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// RestoreObjectAsync.</param>
    /// <param name="bucketName">A string representing the name of the
    /// bucket where the object was located before it was archived.</param>
    /// <param name="objectKey">A string representing the name of the
    /// archived object to restore.</param>
    public static async Task RestoreObjectAsync(IAmazonS3 client, string
bucketName, string objectKey)
    {
        try
        {
            var restoreRequest = new RestoreObjectRequest
            {
                BucketName = bucketName,
                Key = objectKey,
                Days = 2,
            };
            RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

            // Check the status of the restoration.
            await CheckRestorationStatusAsync(client, bucketName, objectKey);
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            Console.WriteLine($"Error: {amazonS3Exception.Message}");
        }
    }

```

```

    /// <summary>
    /// This method retrieves the status of the object's restoration.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// GetObjectMetadataAsync.</param>
    /// <param name="bucketName">A string representing the name of the Amazon
    /// S3 bucket which contains the archived object.</param>
    /// <param name="objectKey">A string representing the name of the
    /// archived object you want to restore.</param>
    public static async Task CheckRestorationStatusAsync(IAmazonS3 client,
string bucketName, string objectKey)
    {
        GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
        };

        GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);

        var restStatus = response.RestoreInProgress ? "in-progress" : "finished
or failed";
        Console.WriteLine($"Restoration status: {restStatus}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [RestoreObject](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer une URL présignée

L'exemple de code suivant montre comment créer une URL présignée pour Amazon S3 et charger un objet.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Générez une URL présignée qui peut exécuter une action Amazon S3 pour une durée limitée.

```
using System;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

public class GenPresignedUrl
{
    public static void Main()
    {
        const string bucketName = "amzn-s3-demo-bucket";
        const string objectKey = "sample.txt";

        // Specify how long the presigned URL lasts, in hours
        const double timeoutDuration = 12;

        // Specify the AWS Region of your Amazon S3 bucket. If it is
        // different from the Region defined for the default user,
        // pass the Region to the constructor for the client. For
        // example: new AmazonS3Client(RegionEndpoint.USEast1);

        // If using the Region us-east-1, and server-side encryption with AWS
        // KMS, you must specify Signature Version 4.
        // Region us-east-1 defaults to Signature Version 2 unless explicitly
        // set to Version 4 as shown below.
        // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
        // userguide/UsingAWSSDK.html#specify-signature-version
        // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
        // TAWSConfigsS3.html
        AWSConfigsS3.UseSignatureVersion4 = true;
        IAmazonS3 s3Client = new AmazonS3Client(RegionEndpoint.USEast1);

        string urlString = GeneratePresignedURL(s3Client, bucketName, objectKey,
        timeoutDuration);
    }
}
```

```

        Console.WriteLine($"The generated URL is: {urlString}.");
    }

    /// <summary>
    /// Generate a presigned URL that can be used to access the file named
    /// in the objectKey parameter for the amount of time specified in the
    /// duration parameter.
    /// </summary>
    /// <param name="client">An initialized S3 client object used to call
    /// the GetPresignedUrl method.</param>
    /// <param name="bucketName">The name of the S3 bucket containing the
    /// object for which to create the presigned URL.</param>
    /// <param name="objectKey">The name of the object to access with the
    /// presigned URL.</param>
    /// <param name="duration">The length of time for which the presigned
    /// URL will be valid.</param>
    /// <returns>A string representing the generated presigned URL.</returns>
    public static string GeneratePresignedURL(IAmazonS3 client, string
bucketName, string objectKey, double duration)
    {
        string urlString = string.Empty;
        try
        {
            var request = new GetPreSignedUrlRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Expires = DateTime.UtcNow.AddHours(duration),
            };
            urlString = client.GetPreSignedURL(request);
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error: '{ex.Message}'");
        }

        return urlString;
    }
}

```

Générez une URL présignée et effectuez un chargement en utilisant cette URL.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to upload an object to an Amazon Simple Storage
/// Service (Amazon S3) bucket using a presigned URL. The code first
/// creates a presigned URL and then uses it to upload an object to an
/// Amazon S3 bucket using that URL.
/// </summary>
public class UploadUsingPresignedURL
{
    private static HttpClient httpClient = new HttpClient();

    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "samplefile.txt";
        string filePath = $"source\\{keyName}";

        // Specify how long the signed URL will be valid in hours.
        double timeoutDuration = 12;

        // Specify the AWS Region of your Amazon S3 bucket. If it is
        // different from the Region defined for the default user,
        // pass the Region to the constructor for the client. For
        // example: new AmazonS3Client(RegionEndpoint.USEast1);

        // If using the Region us-east-1, and server-side encryption with AWS
        KMS, you must specify Signature Version 4.
        // Region us-east-1 defaults to Signature Version 2 unless explicitly
        set to Version 4 as shown below.
        // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingAWSSDK.html#specify-signature-version
        // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/TAWSConfigsS3.html
        AWSConfigsS3.UseSignatureVersion4 = true;
        IAmazonS3 client = new AmazonS3Client(RegionEndpoint.USEast1);
```

```

        var url = GeneratePreSignedURL(client, bucketName, keyName,
timeoutDuration);
        var success = await UploadObject(filePath, url);

        if (success)
        {
            Console.WriteLine("Upload succeeded.");
        }
        else
        {
            Console.WriteLine("Upload failed.");
        }
    }

    /// <summary>
    /// Uploads an object to an Amazon S3 bucket using the presigned URL passed
in
    /// the url parameter.
    /// </summary>
    /// <param name="filePath">The path (including file name) to the local
    /// file you want to upload.</param>
    /// <param name="url">The presigned URL that will be used to upload the
    /// file to the Amazon S3 bucket.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// operation, based on the HttpResponseMessage.</returns>
    public static async Task<bool> UploadObject(string filePath, string url)
    {
        using var streamContent = new StreamContent(
            new FileStream(filePath, FileMode.Open, FileAccess.Read));

        var response = await httpClient.PutAsync(url, streamContent);
        return response.IsSuccessStatusCode;
    }

    /// <summary>
    /// Generates a presigned URL which will be used to upload an object to
    /// an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// GetPreSignedURL.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket to which the
    /// presigned URL will point.</param>

```

```
    /// <param name="objectKey">The name of the file that will be uploaded.</param>
    /// <param name="duration">How long (in hours) the presigned URL will be valid.</param>
    /// <returns>The generated URL.</returns>
    public static string GeneratePreSignedURL(
        IAmazonS3 client,
        string bucketName,
        string objectKey,
        double duration)
    {
        var request = new GetPreSignedUrlRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Verb = HttpVerb.PUT,
            Expires = DateTime.UtcNow.AddHours(duration),
        };

        string url = client.GetPreSignedURL(request);
        return url;
    }
}
```

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour .NET

Montre comment utiliser l'API Java Amazon Rekognition afin de créer une application qui, avec Amazon Rekognition, permet d'identifier des objets par catégorie dans des images stockées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Démarrer avec le chiffrement

L'exemple de code suivant montre comment démarrer avec le chiffrement des objets Amazon S3.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to apply client encryption to an object in an
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class SSEClientEncryption
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "exampleobject.txt";
        string copyTargetKeyName = "examplecopy.txt";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();

        try
        {
            // Create an encryption key.
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);

            // Upload the object.
```

```

        PutObjectRequest putObjectRequest = await UploadObjectAsync(client,
bucketName, keyName, base64Key);

        // Download the object and verify that its contents match what you
uploaded.
        await DownloadObjectAsync(client, bucketName, keyName, base64Key,
putObjectRequest);

        // Get object metadata and verify that the object uses AES-256
encryption.
        await GetObjectMetadataAsync(client, bucketName, keyName,
base64Key);

        // Copy both the source and target objects using server-side
encryption with
        // an encryption key.
        await CopyObjectAsync(client, bucketName, keyName,
copyTargetKeyName, aesEncryption, base64Key);
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

/// <summary>
/// Uploads an object to an Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// PutObjectAsync.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket to which the
/// object will be uploaded.</param>
/// <param name="keyName">The name of the object to upload to the Amazon S3
/// bucket.</param>
/// <param name="base64Key">The encryption key.</param>
/// <returns>The PutObjectRequest object for use by DownloadObjectAsync.</
returns>
public static async Task<PutObjectRequest> UploadObjectAsync(
    IAmazonS3 client,
    string bucketName,
    string keyName,
    string base64Key)
{

```



```

        PutObjectRequest putObjectRequest = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = keyName,
            ContentBody = "sample text",
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key,
        };
        PutObjectResponse putObjectResponse = await
client.PutObjectAsync(putObjectRequest);
        return putObjectRequest;
    }

    /// <summary>
    /// Downloads an encrypted object from an Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// GetObjectAsync.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
object
    /// is located.</param>
    /// <param name="keyName">The name of the Amazon S3 object to download.</
param>
    /// <param name="base64Key">The encryption key used to encrypt the
    /// object.</param>
    /// <param name="putObjectRequest">The PutObjectRequest used to upload
    /// the object.</param>
    public static async Task DownloadObjectAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string base64Key,
        PutObjectRequest putObjectRequest)
    {
        GetObjectRequest getObjectRequest = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = keyName,

            // Provide encryption information for the object stored in Amazon
S3.

```

```

        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key,
    };

    using (GetObjectResponse getResponse = await
client.GetObjectAsync(getObjectRequest))
        using (StreamReader reader = new
StreamReader(getResponse.ResponseStream))
        {
            string content = reader.ReadToEnd();
            if (string.Compare(putObjectRequest.ContentBody, content) == 0)
            {
                Console.WriteLine("Object content is same as we uploaded");
            }
            else
            {
                Console.WriteLine("Error...Object content is not same.");
            }

            if (getResponse.ServerSideEncryptionCustomerMethod ==
ServerSideEncryptionCustomerMethod.AES256)
            {
                Console.WriteLine("Object encryption method is AES256, same as
we set");
            }
            else
            {
                Console.WriteLine("Error...Object encryption method is not the
same as AES256 we set");
            }
        }
    }

    /// <summary>
    /// Retrieves the metadata associated with an Amazon S3 object.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used
    /// to call GetObjectMetadataAsync.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket containing the
    /// object for which we want to retrieve metadata.</param>
    /// <param name="keyName">The name of the object for which we wish to
    /// retrieve the metadata.</param>
    /// <param name="base64Key">The encryption key associated with the

```

```

    /// object.</param>
    public static async Task GetObjectMetadataAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string base64Key)
    {
        GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
        {
            BucketName = bucketName,
            Key = keyName,

            // The object stored in Amazon S3 is encrypted, so provide the
necessary encryption information.
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key,
        };

        GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
        Console.WriteLine("The object metadata show encryption method used is:
{0}", getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
    }

    /// <summary>
    /// Copies an encrypted object from one Amazon S3 bucket to another.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// CopyObjectAsync.</param>
    /// <param name="bucketName">The Amazon S3 bucket containing the object
    /// to copy.</param>
    /// <param name="keyName">The name of the object to copy.</param>
    /// <param name="copyTargetKeyName">The Amazon S3 bucket to which the object
    /// will be copied.</param>
    /// <param name="aesEncryption">The encryption type to use.</param>
    /// <param name="base64Key">The encryption key to use.</param>
    public static async Task CopyObjectAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string copyTargetKeyName,

```

```
Aes aesEncryption,
string base64Key)
{
    aesEncryption.GenerateKey();
    string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

    CopyObjectRequest copyRequest = new CopyObjectRequest
    {
        SourceBucket = bucketName,
        SourceKey = keyName,
        DestinationBucket = bucketName,
        DestinationKey = copyTargetKeyName,

        // Information about the source object's encryption.
        CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,

        // Information about the target object's encryption.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = copyBase64Key,
    };
    await client.CopyObjectAsync(copyRequest);
}
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CopyObject](#)
 - [GetObject](#)
 - [GetObjectMetadata](#)

Démarrer avec les étiquettes

L'exemple de code suivant montre comment démarrer avec des étiquettes pour les objets Amazon S3.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to work with tags in Amazon Simple Storage
/// Service (Amazon S3) objects.
/// </summary>
public class ObjectTag
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "newobject.txt";
        string filePath = @"*** file path ***";

        // Specify your bucket region (an example region is shown).
        RegionEndpoint bucketRegion = RegionEndpoint.USSouth1;

        var client = new AmazonS3Client(bucketRegion);
        await PutObjectsWithTagsAsync(client, bucketName, keyName, filePath);
    }

    /// <summary>
    /// This method uploads an object with tags. It then shows the tag
    /// values, changes the tags, and shows the new tags.
    /// </summary>
    /// <param name="client">The Initialized Amazon S3 client object used
    /// to call the methods to create and change an objects tags.</param>
    /// <param name="bucketName">A string representing the name of the
    /// bucket where the object will be stored.</param>
}
```

```

    /// <param name="keyName">A string representing the key name of the
    /// object to be tagged.</param>
    /// <param name="filePath">The directory location and file name of the
    /// object to be uploaded to the Amazon S3 bucket.</param>
    public static async Task PutObjectsWithTagsAsync(IAmazonS3 client, string
bucketName, string keyName, string filePath)
    {
        try
        {
            // Create an object with tags.
            var putRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                FilePath = filePath,
                TagSet = new List<Tag>
                {
                    new Tag { Key = "Keyx1", Value = "Value1" },
                    new Tag { Key = "Keyx2", Value = "Value2" },
                },
            };

            PutObjectResponse response = await
client.PutObjectAsync(putRequest);

            // Now retrieve the new object's tags.
            GetObjectTaggingRequest getTagsRequest = new
GetObjectTaggingRequest()
            {
                BucketName = bucketName,
                Key = keyName,
            };

            GetObjectTaggingResponse objectTags = await
client.GetObjectTaggingAsync(getTagsRequest);

            // Display the tag values.
            objectTags.Tagging
                .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));

            Tagging newTagSet = new Tagging()
            {
                TagSet = new List<Tag>

```

```

        {
            new Tag { Key = "Key3", Value = "Value3" },
            new Tag { Key = "Key4", Value = "Value4" },
        },
    };

    PutObjectTaggingRequest putObjTagsRequest = new
PutObjectTaggingRequest()
    {
        BucketName = bucketName,
        Key = keyName,
        Tagging = newTagSet,
    };

    PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

    // Retrieve the tags again and show the values.
    GetObjectTaggingRequest getTagsRequest2 = new
GetObjectTaggingRequest()
    {
        BucketName = bucketName,
        Key = keyName,
    };
    GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);

    objectTags2.Tagging
        .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine(
            $"Error: '{ex.Message}'");
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [GetObjectTagging](#) à la section Référence des AWS SDK pour .NET API.

Verrouillage d'objets Amazon S3

L'exemple de code suivant montre comment travailler avec les fonctionnalités de verrouillage d'objets S3.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif illustrant les fonctionnalités de verrouillage d'objet d'Amazon S3.

```
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace S3ObjectLockScenario;

public static class S3ObjectLockWorkflow
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. Create test Amazon Simple Storage Service (S3) buckets with different
        lock policies.
        2. Upload sample objects to each bucket.
        3. Set some Legal Hold and Retention Periods on objects and buckets.
        4. Investigate lock policies by viewing settings or attempting to delete or
        overwrite objects.
        5. Clean up objects and buckets.
    */
}
```



```

public static S3ActionsWrapper _s3ActionsWrapper = null!;
public static IConfiguration _configuration = null!;
private static string _resourcePrefix = null!;
private static string noLockBucketName = null!;
private static string lockEnabledBucketName = null!;
private static string retentionAfterCreationBucketName = null!;
private static List<string> bucketNames = new List<string>();
private static List<string> fileNames = new List<string>();

public static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonS3>()
                .AddTransient<S3ActionsWrapper>()
            )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    ConfigurationSetup();

    ServicesSetup(host);

    try
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon Simple Storage Service (S3)
Object Locking Feature Scenario.");
        Console.WriteLine(new string('-', 80));
        await Setup(true);

        await DemoActionChoices();
    }
}

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cleaning up resources.");
        Console.WriteLine(new string('-', 80));
        await Cleanup(true);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Amazon S3 Object Locking Scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem: {ex.Message}");
        await Cleanup(true);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _s3ActionsWrapper = host.Services.GetRequiredService<S3ActionsWrapper>();
}

/// <summary>
/// Any setup operations needed.
/// </summary>
public static void ConfigurationSetup()
{
    _resourcePrefix = _configuration["resourcePrefix"] ?? "dotnet-example";

    noLockBucketName = _resourcePrefix + "-no-lock";
    lockEnabledBucketName = _resourcePrefix + "-lock-enabled";
    retentionAfterCreationBucketName = _resourcePrefix + "-retention-after-
creation";

    bucketNames.Add(noLockBucketName);
    bucketNames.Add(lockEnabledBucketName);
    bucketNames.Add(retentionAfterCreationBucketName);
}

```

```

// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Setup(bool interactive)
{
    Console.WriteLine(
        "\nFor this scenario, we will use the AWS SDK for .NET to create several
S3\n" +
        "buckets and files to demonstrate working with S3 locking features.\n");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Press Enter when you are ready to start.");
    if (interactive)
        Console.ReadLine();

    Console.WriteLine("\nS3 buckets can be created either with or without object
lock enabled.");
    await _s3ActionsWrapper.CreateBucketWithObjectLock(noLockBucketName, false);
    await _s3ActionsWrapper.CreateBucketWithObjectLock(lockEnabledBucketName,
true);
    await
_s3ActionsWrapper.CreateBucketWithObjectLock(retentionAfterCreationBucketName,
false);

    Console.WriteLine("Press Enter to continue.");
    if (interactive)
        Console.ReadLine();

    Console.WriteLine("\nA bucket can be configured to use object locking with a
default retention period.");
    await
_s3ActionsWrapper.ModifyBucketDefaultRetention(retentionAfterCreationBucketName,
true,
        ObjectLockRetentionMode.Governance, DateTime.UtcNow.AddDays(1));

    Console.WriteLine("Press Enter to continue.");
    if (interactive)
        Console.ReadLine();

    Console.WriteLine("\nObject lock policies can also be added to existing
buckets.");

```

```

        await _s3ActionsWrapper.EnableObjectLockOnBucket(lockEnabledBucketName);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        // Upload some files to the buckets.
        Console.WriteLine("\nNow let's add some test files:");
        var fileName = _configuration["exampleFileName"] ?? "exampleFile.txt";
        int fileCount = 2;
        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for uploading to a bucket.");
        }

        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileCount; i++)
            {
                var numberedFileName = Path.GetFileNameWithoutExtension(fileName) +
i + Path.GetExtension(fileName);
                fileNames.Add(numberedFileName);
                await _s3ActionsWrapper.UploadFileAsync(bucketName,
numberedFileName, fileName);
            }
        }
        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        if (!interactive)
            return true;
        Console.WriteLine("\nNow we can set some object lock policies on individual
files:");
        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileNames.Count; i++)
            {
                // No modifications to the objects in the first bucket.
                if (bucketName != bucketNames[0])
                {

```

```

        var exampleFileName = fileNames[i];
        switch (i)
        {
            case 0:
            {
                var question =
                    $"\\nWould you like to add a legal hold to
{exampleFileName} in {bucketName}? (y/n)";
                if (GetYesNoResponse(question))
                {
                    // Set a legal hold.
                    await
_s3ActionsWrapper.ModifyObjectLegalHold(bucketName, exampleFileName,
ObjectLockLegalHoldStatus.On);

                }
                break;
            }
            case 1:
            {
                var question =
                    $"\\nWould you like to add a 1 day Governance
retention period to {exampleFileName} in {bucketName}? (y/n)" +
                    "\\nReminder: Only a user with the
s3:BypassGovernanceRetention permission will be able to delete this file or its
bucket until the retention period has expired.";
                if (GetYesNoResponse(question))
                {
                    // Set a Governance mode retention period for 1
day.
                    await
_s3ActionsWrapper.ModifyObjectRetentionPeriod(
                        bucketName, exampleFileName,
                        ObjectLockRetentionMode.Governance,
                        DateTime.UtcNow.AddDays(1));
                }
                break;
            }
        }
    }
}
}
Console.WriteLine(new string('-', 80));
return true;

```

```

    }

    /// <summary>
    /// List all of the current buckets and objects.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>The list of buckets and objects.</returns>
    public static async Task<List<S3ObjectVersion>> ListBucketsAndObjects(bool
interactive)
    {
        var allObjects = new List<S3ObjectVersion>();
        foreach (var bucketName in bucketNames)
        {
            var objectsInBucket = await
_s3ActionsWrapper.ListBucketObjectsAndVersions(bucketName);
            foreach (var objectKey in objectsInBucket.Versions)
            {
                allObjects.Add(objectKey);
            }
        }

        if (interactive)
        {
            Console.WriteLine("\nCurrent buckets and objects:\n");
            int i = 0;
            foreach (var bucketObject in allObjects)
            {
                i++;
                Console.WriteLine(
                    $"{i}: {bucketObject.Key} \n\tBucket:
{bucketObject.BucketName}\n\tVersion: {bucketObject.VersionId}");
            }
        }

        return allObjects;
    }

    /// <summary>
    /// Present the user with the demo action choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<bool> DemoActionChoices()
    {
        var choices = new string[]{

```

```

        "List all files in buckets.",
        "Attempt to delete a file.",
        "Attempt to delete a file with retention period bypass.",
        "Attempt to overwrite a file.",
        "View the object and bucket retention settings for a file.",
        "View the legal hold settings for a file.",
        "Finish the scenario."};

var choice = 0;
// Keep asking the user until they choose to move on.
while (choice != 6)
{
    Console.WriteLine(new string('-', 80));
    choice = GetChoiceResponse(
        "\nExplore the S3 locking features by selecting one of the following
choices:"
        , choices);
    Console.WriteLine(new string('-', 80));
    switch (choice)
    {
        case 0:
        {
            await ListBucketsAndObjects(true);
            break;
        }
        case 1:
        {
            Console.WriteLine("\nEnter the number of the object to
delete:");

            var allFiles = await ListBucketsAndObjects(true);
            var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());

            await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, false, allFiles[fileChoice].VersionId);
            break;
        }
        case 2:
        {
            Console.WriteLine("\nEnter the number of the object to
delete:");

            var allFiles = await ListBucketsAndObjects(true);
            var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());

```

```

        await
        _s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, true, allFiles[fileChoice].VersionId);
        break;
    }
    case 3:
    {
        var allFiles = await ListBucketsAndObjects(true);
        Console.WriteLine("\nEnter the number of the object to
overwrite:");
        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
        // Create the file if it does not already exist.
        if (!File.Exists(allFiles[fileChoice].Key))
        {
            await using StreamWriter sw =
File.CreateText(allFiles[fileChoice].Key);
            await sw.WriteLineAsync(
                "This is a sample file for uploading to a bucket.");
        }
        await
        _s3ActionsWrapper.UploadFileAsync(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, allFiles[fileChoice].Key);
        break;
    }
    case 4:
    {
        var allFiles = await ListBucketsAndObjects(true);
        Console.WriteLine("\nEnter the number of the object and
bucket to view:");
        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
        await
        _s3ActionsWrapper.GetObjectRetention(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
        await
        _s3ActionsWrapper.GetBucketObjectLockConfiguration(allFiles[fileChoice].BucketName);
        break;
    }
    case 5:
    {
        var allFiles = await ListBucketsAndObjects(true);
        Console.WriteLine("\nEnter the number of the object to
view:");

```



```

        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
        await
_s3ActionsWrapper.GetObjectLegalHold(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
        break;
    }
}
return true;
}

// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Cleanup(bool interactive)
{
    Console.WriteLine(new string('-', 80));

    if (!interactive || GetYesNoResponse("Do you want to clean up all files and
buckets? (y/n) "))
    {
        // Remove all locks and delete all buckets and objects.
        var allFiles = await ListBucketsAndObjects(false);
        foreach (var fileInfo in allFiles)
        {
            // Check for a legal hold.
            var legalHold = await
_s3ActionsWrapper.GetObjectLegalHold(fileInfo.BucketName, fileInfo.Key);
            if (legalHold?.Status?.Value == ObjectLockLegalHoldStatus.On)
            {
                await
_s3ActionsWrapper.ModifyObjectLegalHold(fileInfo.BucketName, fileInfo.Key,
ObjectLockLegalHoldStatus.Off);
            }

            // Check for a retention period.
            var retention = await
_s3ActionsWrapper.GetObjectRetention(fileInfo.BucketName, fileInfo.Key);
            var hasRetentionPeriod = retention?.Mode ==
ObjectLockRetentionMode.Governance && retention.RetainUntilDate >
DateTime.UtcNow.Date;

```

```

        await _s3ActionsWrapper.DeleteObjectFromBucket(fileInfo.BucketName,
fileInfo.Key, hasRetentionPeriod, fileInfo.VersionId);
    }

    foreach (var bucketName in bucketNames)
    {
        await _s3ActionsWrapper.DeleteBucketByName(bucketName);
    }

}
else
{
    Console.WriteLine(
        "Ok, we'll leave the resources intact.\n" +
        "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
    );
}

Console.WriteLine(new string('-', 80));
return true;
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
    return response;
}

/// <summary>
/// Helper method to get a choice response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="choices">The choices to print on the console.</param>
/// <returns>The index of the selected choice</returns>
private static int GetChoiceResponse(string? question, string[] choices)

```

```

    {
        if (question != null)
        {
            Console.WriteLine(question);

            for (int i = 0; i < choices.Length; i++)
            {
                Console.WriteLine($"{i + 1}. {choices[i]}");
            }
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > choices.Length)
        {
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        return choiceNumber - 1;
    }
}

```

Une classe encapsuleur pour les fonctions S3.

```

using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

namespace S3ObjectLockScenario;

/// <summary>
/// Encapsulate the Amazon S3 operations.
/// </summary>
public class S3ActionsWrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Constructor for the S3ActionsWrapper.
    /// </summary>

```

```

    /// <param name="amazonS3">The injected S3 client.</param>
    public S3ActionsWrapper(IAmazonS3 amazonS3, IConfiguration configuration)
    {
        _amazonS3 = amazonS3;
    }

    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
enableObjectLock)
    {
        Console.WriteLine($"\\tCreating bucket {bucketName} with object lock
{enableObjectLock}.");
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
                ObjectLockEnabledForBucket = enableObjectLock,
            };

            var response = await _amazonS3.PutBucketAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Enable object lock on an existing bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to modify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableObjectLockOnBucket(string bucketName)

```

```

    {
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
                },
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\\tAdded an object lock policy to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error modifying object lock: '{ex.Message}');
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date retention expires.</param>
    /// <returns>True if successful.</returns>

```

```

    public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
        string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
    {
        try
        {
            var request = new PutObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Retention = new ObjectLockRetention()
                {
                    Mode = retention,
                    RetainUntilDate = retainUntilDate
                }
            };

            var response = await _amazonS3.PutObjectRetentionAsync(request);
            Console.WriteLine($"{\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}."");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"{\tError modifying retention period:
'{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to modify.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date for retention until.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
    {
        var enabledString = enableObjectLock ? "Enabled" : "Disabled";
        var timeDifference = retainUntilDate.Subtract(DateTime.Now);
        try
        {

```

```

        // First, enable Versioning on the bucket.
        await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
        {
            BucketName = bucketName,
            VersioningConfig = new S3BucketVersioningConfig()
            {
                EnableMfaDelete = false,
                Status = VersionStatus.Enabled
            }
        });

        var request = new PutObjectLockConfigurationRequest()
        {
            BucketName = bucketName,
            ObjectLockConfiguration = new ObjectLockConfiguration()
            {
                ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                Rule = new ObjectLockRule()
                {
                    DefaultRetention = new DefaultRetention()
                    {
                        Mode = retention,
                        Days = timeDifference.Days // Can be specified in days
or years but not both.
                    }
                }
            }
        };

        var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
        Console.WriteLine($"\\tAdded a default retention to bucket
{bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tError modifying object lock: '{ex.Message}'");
        return false;
    }
}

/// <summary>
/// Get the retention period for an S3 object.

```

```

    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object retention details.</returns>
    public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectRetentionAsync(request);
            Console.WriteLine($"{Environment.NewLine}Object retention for {objectKey} in {bucketName}:
" +
                                Environment.NewLine}{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}");
            return response.Retention;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"{Environment.NewLine}Unable to fetch object lock retention:
'{ex.Message}'");
            return new ObjectLockRetention();
        }
    }

    /// <summary>
    /// Set or modify a legal hold on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="holdStatus">The On or Off status for the legal hold.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectLegalHold(string bucketName,
        string objectKey, ObjectLockLegalHoldStatus holdStatus)
    {
        try
        {
            var request = new PutObjectLegalHoldRequest()
            {

```



```

        BucketName = bucketName,
        Key = objectKey,
        LegalHold = new ObjectLockLegalHold()
        {
            Status = holdStatus
        }
    };

    var response = await _amazonS3.PutObjectLegalHoldAsync(request);
    Console.WriteLine($"\\tModified legal hold for {objectKey} in
{bucketName}.");
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"\\tError modifying legal hold: '{ex.Message}'");
    return false;
}
}

/// <summary>
/// Get the legal hold details for an S3 object.
/// </summary>
/// <param name="bucketName">The bucket of the object.</param>
/// <param name="objectKey">The object key.</param>
/// <returns>The object legal hold details.</returns>
public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
    string objectKey)
{
    try
    {
        var request = new GetObjectLegalHoldRequest()
        {
            BucketName = bucketName,
            Key = objectKey
        };

        var response = await _amazonS3.GetObjectLegalHoldAsync(request);
        Console.WriteLine($"\\tObject legal hold for {objectKey} in {bucketName}:
" +
            $"\\n\\tStatus: {response.LegalHold.Status}");
        return response.LegalHold;
    }
    catch (AmazonS3Exception ex)

```

```

        {
            Console.WriteLine($"Unable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }

    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>
    /// <returns>The bucket's object lock configuration details.</returns>
    public async Task<ObjectLockConfiguration>
    GetBucketObjectLockConfiguration(string bucketName)
    {
        try
        {
            var request = new GetObjectLockConfigurationRequest()
            {
                BucketName = bucketName
            };

            var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
            Console.WriteLine($"Bucket object lock config for {bucketName} in
{bucketName}: " +
                $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

            return response.ObjectLockConfiguration;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Unable to fetch object lock config:
'{ex.Message}'");
            return new ObjectLockConfiguration();
        }
    }

    /// <summary>
    /// Upload a file from the local computer to an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectName">The object to upload.</param>

```

```

    /// <param name="filePath">The path, including file name, of the object to
upload.</param>
    /// <returns>True if success.</returns>
    public async Task<bool> UploadFileAsync(string bucketName, string objectName,
string filePath)
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
            ChecksumAlgorithm = ChecksumAlgorithm.SHA256
        };

        var response = await _amazonS3.PutObjectAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"\\tSuccessfully uploaded {objectName} to
{bucketName}.");
            return true;
        }
        else
        {
            Console.WriteLine($"\\tCould not upload {objectName} to {bucketName}.");
            return false;
        }
    }

    /// <summary>
    /// List bucket objects and versions.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <returns>The list of objects and versions.</returns>
    public async Task<ListVersionsResponse> ListBucketObjectsAndVersions(string
bucketName)
    {
        var request = new ListVersionsRequest()
        {
            BucketName = bucketName
        };

        var response = await _amazonS3.ListVersionsAsync(request);
        return response;
    }

```

```

    /// <summary>
    /// Delete an object from a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="hasRetention">True if the object has retention settings.</
param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteObjectFromBucket(string bucketName, string
objectKey, bool hasRetention, string? versionId = null)
    {
        try
        {
            var request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                VersionId = versionId,
            };
            if (hasRetention)
            {
                // Set the BypassGovernanceRetention header
                // if the file has retention settings.
                request.BypassGovernanceRetention = true;
            }
            await _amazonS3.DeleteObjectAsync(request);
            Console.WriteLine(
                $"Deleted {objectKey} in {bucketName}.");
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Unable to delete object {objectKey} in bucket
{bucketName}: " + ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Delete a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>

```

```
/// <param name="objectKey">The key of the object to delete.</param>
/// <param name="versionId">Optional versionId.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteBucketByName(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest() { BucketName = bucketName, };
        var response = await _amazonS3.DeleteBucketAsync(request);
        Console.WriteLine($"\\tDelete for {bucketName} complete.");
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"\\tUnable to delete bucket {bucketName}: " +
ex.Message);
        return false;
    }
}
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetention](#)
 - [PutObjectLegalHold](#)
 - [PutObjectLockConfiguration](#)
 - [PutObjectRetention](#)

Réalisation de demandes conditionnelles

L'exemple de code suivant montre comment ajouter des conditions préalables aux demandes Amazon S3.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif illustrant les fonctionnalités de demande conditionnelle d'Amazon S3.

```
using Amazon.S3;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace S3ConditionalRequestsScenario;

public static class S3ConditionalRequestsScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This example demonstrates the use of conditional requests for S3 operations.
        You can use conditional requests to add preconditions to S3 read requests to
        return or copy
        an object based on its Entity tag (ETag), or last modified date.
        You can use a conditional write requests to prevent overwrites by ensuring
        there is no existing object with the same key.
    */

    public static S3ActionsWrapper _s3ActionsWrapper = null!;
    public static IConfiguration _configuration = null!;
    public static string _resourcePrefix = null!;
    public static string _sourceBucketName = null!;
    public static string _destinationBucketName = null!;
    public static string _sampleObjectKey = null!;
```

```

public static string _sampleObjectEtag = null!;
public static bool _interactive = true;

public static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonS3>()
                .AddTransient<S3ActionsWrapper>()
        )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    ServicesSetup(host);

    try
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon Simple Storage Service (S3)
Conditional Requests Feature Scenario.");
        Console.WriteLine(new string('-', 80));
        ConfigurationSetup();
        _sampleObjectEtag = await Setup(_sourceBucketName,
            _destinationBucketName, _sampleObjectKey);

        await DisplayDemoChoices(_sourceBucketName, _destinationBucketName,
            _sampleObjectKey, _sampleObjectEtag, 0);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cleaning up resources.");
        Console.WriteLine(new string('-', 80));
    }
}

```

```

        await Cleanup(true);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Amazon S3 Conditional Requests Feature Scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem: {ex.Message}");
        await CleanupScenario(_sourceBucketName, _destinationBucketName);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _s3ActionsWrapper = host.Services.GetRequiredService<S3ActionsWrapper>();
}

/// <summary>
/// Any setup operations needed.
/// </summary>
public static void ConfigurationSetup()
{
    _resourcePrefix = _configuration["resourcePrefix"] ?? "dotnet-example";

    _sourceBucketName = _resourcePrefix + "-source";
    _destinationBucketName = _resourcePrefix + "-dest";
    _sampleObjectKey = _resourcePrefix + "-sample-object.txt";
}

/// <summary>
/// Sets up the scenario by creating a source and destination bucket, and
uploading a test file to the source bucket.
/// </summary>
/// <param name="sourceBucket">The name of the source bucket.</param>
/// <param name="destBucket">The name of the destination bucket.</param>

```



```

    /// <param name="objectKey">The name of the test file to add to the source
    bucket.</param>
    /// <returns>The ETag of the uploaded test file.</returns>
    public static async Task<string> Setup(string sourceBucket, string destBucket,
    string objectKey)
    {
        Console.WriteLine(
            "\nFor this scenario, we will use the AWS SDK for .NET to create several
S3\n" +
            "buckets and files to demonstrate working with S3 conditional requests.
\n" +
            "This example demonstrates the use of conditional requests for S3
operations.\r\n" +
            "You can use conditional requests to add preconditions to S3 read
requests to return or copy\r\n" +
            "an object based on its Entity tag (ETag), or last modified date. \r\n"
+
            "You can use a conditional write requests to prevent overwrites by
ensuring \r\n" +
            "there is no existing object with the same key. \r\n\r\n" +
            "This example will allow you to perform conditional reads\r\n" +
            "and writes that will succeed or fail based on your selected options.\r
\n\r\n" +
            "Sample buckets and a sample object will be created as part of the
example.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you are ready to start.");
        if (_interactive)
            Console.ReadLine();

        await _s3ActionsWrapper.CreateBucketWithName(sourceBucket);
        await _s3ActionsWrapper.CreateBucketWithName(destBucket);

        var eTag = await _s3ActionsWrapper.PutObjectConditional(objectKey,
sourceBucket,
            "Test file content.");

        return eTag;
    }

    /// <summary>
    /// Cleans up the scenario by deleting the source and destination buckets.
    /// </summary>

```

```

    /// <param name="sourceBucket">The name of the source bucket.</param>
    /// <param name="destBucket">The name of the destination bucket.</param>
    public static async Task CleanupScenario(string sourceBucket, string destBucket)
    {
        await _s3ActionsWrapper.CleanupBucketByName(sourceBucket);
        await _s3ActionsWrapper.CleanupBucketByName(destBucket);
    }

    /// <summary>
    /// Displays a list of the objects in the test buckets.
    /// </summary>
    /// <param name="sourceBucket">The name of the source bucket.</param>
    /// <param name="destBucket">The name of the destination bucket.</param>
    public static async Task DisplayBuckets(string sourceBucket, string destBucket)
    {
        await _s3ActionsWrapper.ListBucketContentsByName(sourceBucket);
        await _s3ActionsWrapper.ListBucketContentsByName(destBucket);
    }

    /// <summary>
    /// Displays the menu of conditional request options for the user.
    /// </summary>
    /// <param name="sourceBucket">The name of the source bucket.</param>
    /// <param name="destBucket">The name of the destination bucket.</param>
    /// <param name="objectKey">The key of the test object in the source bucket.</
param>
    /// <param name="etag">The ETag of the test object in the source bucket.</param>
    public static async Task DisplayDemoChoices(string sourceBucket, string
destBucket, string objectKey, string etag, int defaultChoice)
    {
        var actions = new[]
        {
            "Print a list of bucket items.",
            "Perform a conditional read.",
            "Perform a conditional copy.",
            "Perform a conditional write.",
            "Clean up and exit."
        };

        var conditions = new[]
        {
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",

```

```

        "If-Modified-Since: using yesterday's date. This condition should
succeed.",
        "If-Unmodified-Since: using yesterday's date. This condition should
fail."
    };

    var conditionTypes = new[]
    {
        S3ConditionType.IfMatch,
        S3ConditionType.IfNoneMatch,
        S3ConditionType.IfModifiedSince,
        S3ConditionType.IfUnmodifiedSince,
    };

    var yesterdayDate = DateTime.UtcNow.AddDays(-1);

    int choice;
    while ((choice = GetChoiceResponse("\nExplore the S3 conditional request
features by selecting one of the following choices:", actions, defaultChoice)) !=
4)
    {
        switch (choice)
        {
            case 0:
                Console.WriteLine("Listing the objects and buckets.");
                await DisplayBuckets(sourceBucket, destBucket);
                break;
            case 1:
                int conditionTypeIndex = GetChoiceResponse("Perform a
conditional read:", conditions, 1);
                if (conditionTypeIndex == 0 || conditionTypeIndex == 1)
                {
                    await _s3ActionsWrapper.GetObjectConditional(objectKey,
sourceBucket, conditionTypes[conditionTypeIndex], null, _sampleObjectEtag);
                }
                else if (conditionTypeIndex == 2 || conditionTypeIndex == 3)
                {
                    await _s3ActionsWrapper.GetObjectConditional(objectKey,
sourceBucket, conditionTypes[conditionTypeIndex], yesterdayDate);
                }
                break;
            case 2:
                int copyConditionTypeIndex = GetChoiceResponse("Perform a
conditional copy:", conditions, 1);

```

```

        string destKey = GetStringResponse("Enter an object key:",
"sampleObjectKey");
        if (copyConditionTypeIndex == 0 || copyConditionTypeIndex == 1)
        {
            await _s3ActionsWrapper.CopyObjectConditional(objectKey,
destKey, sourceBucket, destBucket, conditionTypes[copyConditionTypeIndex], null,
etag);
        }
        else if (copyConditionTypeIndex == 2 || copyConditionTypeIndex
== 3)
        {
            await _s3ActionsWrapper.CopyObjectConditional(objectKey,
destKey, sourceBucket, destBucket, conditionTypes[copyConditionTypeIndex],
yesterdayDate);
        }
        break;
    case 3:
        Console.WriteLine("Perform a conditional write using IfNoneMatch
condition on the object key.");
        Console.WriteLine("If the key is a duplicate, the write will
fail.");
        string newObjectKey = GetStringResponse("Enter an object key:",
"newObjectKey");
        await _s3ActionsWrapper.PutObjectConditional(newObjectKey,
sourceBucket, "Conditional write example data.");
        break;
    }

    if (!_interactive)
    {
        break;
    }
}

Console.WriteLine("Proceeding to cleanup.");
}

// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Cleanup(bool interactive)
{

```

```

        Console.WriteLine(new string('-', 80));

        if (!interactive || GetYesNoResponse("Do you want to clean up all files and
buckets? (y/n) "))
        {
            await _s3ActionsWrapper.CleanUpBucketByName(_sourceBucketName);
            await _s3ActionsWrapper.CleanUpBucketByName(_destinationBucketName);

        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Helper method to get a choice response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="choices">The choices to print on the console.</param>
    /// <returns>The index of the selected choice</returns>
    private static int GetChoiceResponse(string? question, string[] choices, int
defaultChoice)

```

```

{
    if (question != null)
    {
        Console.WriteLine(question);

        for (int i = 0; i < choices.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {choices[i]}");
        }
    }

    if (!_interactive)
        return defaultChoice;

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > choices.Length)
    {
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }

    return choiceNumber - 1;
}

/// <summary>
/// Get a string response from the user.
/// </summary>
/// <param name="question">The question to print.</param>
/// <param name="defaultAnswer">A default answer to use when not interactive.</
param>
/// <returns>The string response.</returns>
public static string GetStringResponse(string? question, string defaultAnswer)
{
    string? answer = "";
    if (!_interactive)
    {
        do
        {
            Console.WriteLine(question);
            answer = Console.ReadLine();
        } while (string.IsNullOrEmpty(answer));
    }
    else
    {

```

```
        answer = defaultAnswer;
    }

    return answer;
}
}
```

Une classe d'encapsuleur pour les fonctions S3.

```
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Logging;

namespace S3ConditionalRequestsScenario;

/// <summary>
/// Encapsulate the Amazon S3 operations.
/// </summary>
public class S3ActionsWrapper
{
    private readonly IAmazonS3 _amazonS3;
    private readonly ILogger<S3ActionsWrapper> _logger;

    /// <summary>
    /// Constructor for the S3ActionsWrapper.
    /// </summary>
    /// <param name="amazonS3">The injected S3 client.</param>
    /// <param name="logger">The class logger.</param>
    public S3ActionsWrapper(IAmazonS3 amazonS3, ILogger<S3ActionsWrapper> logger)
    {
        _amazonS3 = amazonS3;
        _logger = logger;
    }

    /// <summary>
    /// Retrieves an object from Amazon S3 with a conditional request.
    /// </summary>
    /// <param name="objectKey">The key of the object to retrieve.</param>
    /// <param name="sourceBucket">The source bucket of the object.</param>
```

```

    /// <param name="conditionType">The type of condition: 'IfMatch', 'IfNoneMatch',
    'IfModifiedSince', 'IfUnmodifiedSince'.</param>
    /// <param name="conditionDateValue">The value to use for the condition for
    dates.</param>
    /// <param name="etagConditionalValue">The value to use for the condition for
    etags.</param>
    /// <returns>True if the conditional read is successful, False otherwise.</
returns>
    public async Task<bool> GetObjectConditional(string objectKey, string
sourceBucket,
        S3ConditionType conditionType, DateTime? conditionDateValue = null, string?
etagConditionalValue = null)
    {
        try
        {
            var getObjectRequest = new GetObjectRequest
            {
                BucketName = sourceBucket,
                Key = objectKey
            };

            switch (conditionType)
            {
                case S3ConditionType.IfMatch:
                    getObjectRequest.EtagToMatch = etagConditionalValue;
                    break;
                case S3ConditionType.IfNoneMatch:
                    getObjectRequest.EtagToNotMatch = etagConditionalValue;
                    break;
                case S3ConditionType.IfModifiedSince:
                    getObjectRequest.ModifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
                    break;
                case S3ConditionType.IfUnmodifiedSince:
                    getObjectRequest.UnmodifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
                    break;
                default:
                    throw new ArgumentOutOfRangeException(nameof(conditionType),
conditionType, null);
            }

            var response = await _amazonS3.GetObjectAsync(getObjectRequest);
            var sampleBytes = new byte[20];

```



```

        await response.ResponseStream.ReadAsync(sampleBytes, 0, 20);
        _logger.LogInformation($"Conditional read successful. Here are the first
20 bytes of the object:\n{System.Text.Encoding.UTF8.GetString(sampleBytes)}");
        return true;
    }
    catch (AmazonS3Exception e)
    {
        if (e.ErrorCode == "PreconditionFailed")
        {
            _logger.LogError("Conditional read failed: Precondition failed");
        }
        else if (e.ErrorCode == "NotModified")
        {
            _logger.LogError("Conditional read failed: Object not modified");
        }
        else
        {
            _logger.LogError($"Unexpected error: {e.ErrorCode}");
            throw;
        }
        return false;
    }
}

/// <summary>
/// Uploads an object to Amazon S3 with a conditional request. Prevents
overwrite using an IfNoneMatch condition for the object key.
/// </summary>
/// <param name="objectKey">The key of the object to upload.</param>
/// <param name="bucket">The source bucket of the object.</param>
/// <param name="content">The content to upload as a string.</param>
/// <returns>The ETag if the conditional write is successful, empty otherwise.</
returns>
    public async Task<string> PutObjectConditional(string objectKey, string bucket,
string content)
    {
        try
        {
            var putObjectRequest = new PutObjectRequest
            {
                BucketName = bucket,
                Key = objectKey,
                ContentBody = content,
                IfNoneMatch = "*"
            }
        }
    }

```

```

        };

        var putResult = await _amazonS3.PutObjectAsync(putObjectRequest);
        _logger.LogInformation($"Conditional write successful for key
{objectKey} in bucket {bucket}.");
        return putResult.ETag;
    }
    catch (AmazonS3Exception e)
    {
        if (e.ErrorCode == "PreconditionFailed")
        {
            _logger.LogError("Conditional write failed: Precondition failed");
        }
        else
        {
            _logger.LogError($"Unexpected error: {e.ErrorCode}");
            throw;
        }
        return string.Empty;
    }
}

/// <summary>
/// Copies an object from one Amazon S3 bucket to another with a conditional
request.
/// </summary>
/// <param name="sourceKey">The key of the source object to copy.</param>
/// <param name="destKey">The key of the destination object.</param>
/// <param name="sourceBucket">The source bucket of the object.</param>
/// <param name="destBucket">The destination bucket of the object.</param>
/// <param name="conditionType">The type of condition to apply, e.g.
'CopySourceIfMatch', 'CopySourceIfNoneMatch', 'CopySourceIfModifiedSince',
'CopySourceIfUnmodifiedSince'.</param>
/// <param name="conditionDateValue">The value to use for the condition for
dates.</param>
/// <param name="etagConditionalValue">The value to use for the condition for
etags.</param>
/// <returns>True if the conditional copy is successful, False otherwise.</
returns>
public async Task<bool> CopyObjectConditional(string sourceKey, string destKey,
string sourceBucket, string destBucket,
S3ConditionType conditionType, DateTime? conditionDateValue = null, string?
etagConditionalValue = null)
{

```

```

try
{
    var copyObjectRequest = new CopyObjectRequest
    {
        DestinationBucket = destBucket,
        DestinationKey = destKey,
        SourceBucket = sourceBucket,
        SourceKey = sourceKey
    };

    switch (conditionType)
    {
        case S3ConditionType.IfMatch:
            copyObjectRequest.ETagToMatch = etagConditionalValue;
            break;
        case S3ConditionType.IfNoneMatch:
            copyObjectRequest.ETagToNotMatch = etagConditionalValue;
            break;
        case S3ConditionType.IfModifiedSince:
            copyObjectRequest.ModifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
            break;
        case S3ConditionType.IfUnmodifiedSince:
            copyObjectRequest.UnmodifiedSinceDateUtc =
conditionDateValue.GetValueOrDefault();
            break;
        default:
            throw new ArgumentOutOfRangeException(nameof(conditionType),
conditionType, null);
    }

    await _amazonS3.CopyObjectAsync(copyObjectRequest);
    _logger.LogInformation($"Conditional copy successful for key {destKey}
in bucket {destBucket}.");
    return true;
}
catch (AmazonS3Exception e)
{
    if (e.ErrorCode == "PreconditionFailed")
    {
        _logger.LogError("Conditional copy failed: Precondition failed");
    }
    else if (e.ErrorCode == "304")
    {

```

```

        _logger.LogError("Conditional copy failed: Object not modified");
    }
    else
    {
        _logger.LogError($"Unexpected error: {e.ErrorCode}");
        throw;
    }
    return false;
}
}

/// <summary>
/// Create a new Amazon S3 bucket with a specified name and check that the
bucket is ready.
/// </summary>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateBucketWithName(string bucketName)
{
    Console.WriteLine($"\\tCreating bucket {bucketName}.");
    try
    {
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true
            };

            await _amazonS3.PutBucketAsync(request);
            var bucketReady = false;
            var retries = 5;
            while (!bucketReady && retries > 0)
            {
                Thread.Sleep(5000);
                bucketReady = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_amazonS3, bucketName);
                retries--;
            }

            return bucketReady;
        }
        catch (BucketAlreadyExistsException ex)
        {
            Console.WriteLine($"Bucket already exists: '{ex.Message}'");

```

```

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error creating bucket: '{ex.Message}'");
        return false;
    }
}

/// <summary>
/// Cleans up objects and deletes the bucket by name.
/// </summary>
/// <param name="bucketName">The name of the bucket.</param>
/// <returns>Async task.</returns>
public async Task CleanupBucketByName(string bucketName)
{
    try
    {
        var listObjectsResponse = await _amazonS3.ListObjectsV2Async(new
ListObjectsV2Request { BucketName = bucketName });
        foreach (var obj in listObjectsResponse.S3Objects)
        {
            await _amazonS3.DeleteObjectAsync(new DeleteObjectRequest
{ BucketName = bucketName, Key = obj.Key });
        }
        await _amazonS3.DeleteBucketAsync(new DeleteBucketRequest { BucketName =
bucketName });
        Console.WriteLine($"Cleaned up bucket: {bucketName}.");
    }
    catch (AmazonS3Exception e)
    {
        if (e.ErrorCode == "NoSuchBucket")
        {
            Console.WriteLine($"Bucket {bucketName} does not exist, skipping
cleanup.");
        }
        else
        {
            Console.WriteLine($"Error deleting bucket: {e.ErrorCode}");
            throw;
        }
    }
}
}

```

```
/// <summary>
/// List the contents of the bucket with their ETag.
/// </summary>
/// <param name="bucketName">The name of the bucket.</param>
/// <returns>Async task.</returns>
public async Task<List<S3Object>> ListBucketContentsByName(string bucketName)
{
    var results = new List<S3Object>();
    try
    {
        Console.WriteLine($"\\t Items in bucket {bucketName}");
        var listObjectsResponse = await _amazonS3.ListObjectsV2Async(new
ListObjectsV2Request { BucketName = bucketName });
        if (listObjectsResponse.S3Objects.Count == 0)
        {
            Console.WriteLine("\\t\\tNo objects found.");
        }
        else
        {
            foreach (var obj in listObjectsResponse.S3Objects)
            {
                Console.WriteLine($"\\t\\t object: {obj.Key} ETag {obj.ETag}");
            }
        }
        results = listObjectsResponse.S3Objects;
    }
    catch (AmazonS3Exception e)
    {
        if (e.ErrorCode == "NoSuchBucket")
        {
            _logger.LogError($"Bucket {bucketName} does not exist.");
        }
        else
        {
            _logger.LogError($"Error listing bucket and objects:
{e.ErrorCode}");
            throw;
        }
    }

    return results;
}
```

```

    /// <summary>
    /// Delete an object from a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteObjectFromBucket(string bucketName, string
objectKey)
    {
        try
        {
            var request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };
            await _amazonS3.DeleteObjectAsync(request);
            Console.WriteLine($"Deleted {objectKey} in {bucketName}.");
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Unable to delete object {objectKey} in bucket
{bucketName}: " + ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Delete a specific bucket by deleting the objects and then the bucket itself.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CleanUpBucketByName(string bucketName)
    {
        try
        {
            var allFiles = await ListBucketContentsByName(bucketName);

            foreach (var fileInfo in allFiles)
            {
                await DeleteObjectFromBucket(fileInfo.BucketName, fileInfo.Key);
            }
        }
    }

```

```
    }

    var request = new DeleteBucketRequest() { BucketName = bucketName, };
    var response = await _amazonS3.DeleteBucketAsync(request);
    Console.WriteLine($"\\tDelete for {bucketName} complete.");
    return response.HttpStatusCode == HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"\\tUnable to delete bucket {bucketName}: " +
ex.Message);
    return false;
}
}
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CopyObject](#)
 - [GetObject](#)
 - [PutObject](#)

Gérer les listes de contrôle d'accès (ACLs)

L'exemple de code suivant montre comment gérer les listes de contrôle d'accès (ACLs) pour les compartiments Amazon S3.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
```



```

using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to manage Amazon Simple Storage Service
/// (Amazon S3) access control lists (ACLs) to control Amazon S3 bucket
/// access.
/// </summary>
public class ManageACLs
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket1";
        string newBucketName = "amzn-s3-demo-bucket2";
        string keyName = "sample-object.txt";
        string emailAddress = "someone@example.com";

        // If the AWS Region where your bucket is located is different from
        // the Region defined for the default user, pass the Amazon S3 bucket's
        // name to the client constructor. It should look like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USEast1;
        IAmazonS3 client = new AmazonS3Client();

        await TestBucketObjectACLsAsync(client, bucketName, newBucketName,
keyName, emailAddress);
    }

    /// <summary>
    /// Creates a new Amazon S3 bucket with a canned ACL, then retrieves the ACL
    /// information and then adds a new ACL to one of the objects in the
    /// Amazon S3 bucket.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
call
    /// methods to create a bucket, get an ACL, and add a different ACL to
    /// one of the objects.</param>
    /// <param name="bucketName">A string representing the original Amazon S3
    /// bucket name.</param>
    /// <param name="newBucketName">A string representing the name of the
    /// new bucket that will be created.</param>
    /// <param name="keyName">A string representing the key name of an Amazon S3
    /// object for which we will change the ACL.</param>
    /// <param name="emailAddress">A string representing the email address

```

```

    /// belonging to the person to whom access to the Amazon S3 bucket will be
    /// granted.</param>
    public static async Task TestBucketObjectACLsAsync(
        IAmazonS3 client,
        string bucketName,
        string newBucketName,
        string keyName,
        string emailAddress)
    {
        try
        {
            // Create a new Amazon S3 bucket and specify canned ACL.
            var success = await CreateBucketWithCannedACLAsync(client,
newBucketName);

            // Get the ACL on a bucket.
            await GetBucketACLAsync(client, bucketName);

            // Add (replace) the ACL on an object in a bucket.
            await AddACLToExistingObjectAsync(client, bucketName, keyName,
emailAddress);
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            Console.WriteLine($"Exception: {amazonS3Exception.Message}");
        }
    }

    /// <summary>
    /// Creates a new Amazon S3 bucket with a canned ACL attached.
    /// </summary>
    /// <param name="client">The initialized client object used to call
    /// PutBucketAsync.</param>
    /// <param name="newBucketName">A string representing the name of the
    /// new Amazon S3 bucket.</param>
    /// <returns>Returns a boolean value indicating success or failure.</
returns>
    public static async Task<bool> CreateBucketWithCannedACLAsync(IAmazonS3
client, string newBucketName)
    {
        var request = new PutBucketRequest()
        {
            BucketName = newBucketName,
            BucketRegion = S3Region.EUWest1,

```

```

        // Add a canned ACL.
        CannedACL = S3CannedACL.LogDeliveryWrite,
    };

    var response = await client.PutBucketAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Retrieves the ACL associated with the Amazon S3 bucket name in the
/// bucketName parameter.
/// </summary>
/// <param name="client">The initialized client object used to call
/// PutBucketAsync.</param>
/// <param name="bucketName">The Amazon S3 bucket for which we want to get
the
/// ACL list.</param>
/// <returns>Returns an S3AccessControlList returned from the call to
/// GetACLAsync.</returns>
public static async Task<S3AccessControlList> GetBucketACLAsync(IAmazonS3
client, string bucketName)
{
    GetACLResponse response = await client.GetACLAsync(new GetACLRequest
    {
        BucketName = bucketName,
    });

    return response.AccessControlList;
}

/// <summary>
/// Adds a new ACL to an existing object in the Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized client object used to call
/// PutBucketAsync.</param>
/// <param name="bucketName">A string representing the name of the Amazon S3
/// bucket containing the object to which we want to apply a new ACL.</
param>
/// <param name="keyName">A string representing the name of the object
/// to which we want to apply the new ACL.</param>

```

```
/// <param name="emailAddress">The email address of the person to whom
/// we will be applying to whom access will be granted.</param>
public static async Task AddACLToExistingObjectAsync(IAmazonS3 client,
string bucketName, string keyName, string emailAddress)
{
    // Retrieve the ACL for an object.
    GetACLResponse aclResponse = await client.GetACLAsync(new GetACLRequest
    {
        BucketName = bucketName,
        Key = keyName,
    });

    S3AccessControlList acl = aclResponse.AccessControlList;

    // Retrieve the owner.
    Owner owner = acl.Owner;

    // Clear existing grants.
    acl.Grants.Clear();

    // Add a grant to reset the owner's full permission
    // (the previous clear statement removed all permissions).
    var fullControlGrant = new S3Grant
    {
        Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id },
    };
    acl.AddGrant(fullControlGrant.Grantee, S3Permission.FULL_CONTROL);

    // Specify email to identify grantee for granting permissions.
    var grantUsingEmail = new S3Grant
    {
        Grantee = new S3Grantee { EmailAddress = emailAddress },
        Permission = S3Permission.WRITE_ACP,
    };

    // Specify log delivery group as grantee.
    var grantLogDeliveryGroup = new S3Grant
    {
        Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" },
        Permission = S3Permission.WRITE,
    };

    // Create a new ACL.
```

```
var newAcl = new S3AccessControlList
{
    Grants = new List<S3Grant> { grantUsingEmail,
grantLogDeliveryGroup },
    Owner = owner,
};

// Set the new ACL. We're throwing away the response here.
_ = await client.PutACLAsync(new PutACLRequest
{
    BucketName = bucketName,
    Key = keyName,
    AccessControlList = newAcl,
});
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [GetBucketAcl](#)
 - [GetObjectAcl](#)
 - [PutBucketAcl](#)
 - [PutObjectAcl](#)

Effectuer une copie en plusieurs parties

L'exemple de code suivant montre comment effectuer une copie en plusieurs parties d'un objet Amazon S3.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to perform a multi-part copy from one Amazon
/// Simple Storage Service (Amazon S3) bucket to another.
/// </summary>
public class MPUapiCopyObj
{
    private const string SourceBucket = "amzn-s3-demo-bucket1";
    private const string TargetBucket = "amzn-s3-demo-bucket2";
    private const string SourceObjectKey = "example.mov";
    private const string TargetObjectKey = "copied_video_file.mov";

    /// <summary>
    /// This method starts the multi-part upload.
    /// </summary>
    public static async Task Main()
    {
        var s3Client = new AmazonS3Client();
        Console.WriteLine("Copying object...");
        await MPUCopyObjectAsync(s3Client);
    }

    /// <summary>
    /// This method uses the passed client object to perform a multipart
    /// copy operation.
    /// </summary>
    /// <param name="client">An Amazon S3 client object that will be used
    /// to perform the copy.</param>
    public static async Task MPUCopyObjectAsync(AmazonS3Client client)
    {
        // Create a list to store the copy part responses.
        var copyResponses = new List<CopyPartResponse>();

        // Setup information required to initiate the multipart upload.
        var initiateRequest = new InitiateMultipartUploadRequest
        {
            BucketName = TargetBucket,
            Key = TargetObjectKey,
```

```

};

// Initiate the upload.
InitiateMultipartUploadResponse initResponse =
    await client.InitiateMultipartUploadAsync(initiateRequest);

// Save the upload ID.
string uploadId = initResponse.UploadId;

try
{
    // Get the size of the object.
    var metadataRequest = new GetObjectMetadataRequest
    {
        BucketName = SourceBucket,
        Key = SourceObjectKey,
    };

    GetObjectMetadataResponse metadataResponse =
        await client.GetObjectMetadataAsync(metadataRequest);
    var objectSize = metadataResponse.ContentLength; // Length in bytes.

    // Copy the parts.
    var partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        var copyRequest = new CopyPartRequest
        {
            DestinationBucket = TargetBucket,
            DestinationKey = TargetObjectKey,
            SourceBucket = SourceBucket,
            SourceKey = SourceObjectKey,
            UploadId = uploadId,
            FirstByte = bytePosition,
            LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
            PartNumber = i,
        };

        copyResponses.Add(await client.CopyPartAsync(copyRequest));

        bytePosition += partSize;
    }
}

```

```

    }

    // Set up to complete the copy.
    var completeRequest = new CompleteMultipartUploadRequest
    {
        BucketName = TargetBucket,
        Key = TargetObjectKey,
        UploadId = initResponse.UploadId,
    };
    completeRequest.AddPartETags(copyResponses);

    // Complete the copy.
    CompleteMultipartUploadResponse completeUploadResponse =
        await client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine($"Error encountered on server.
Message: '{e.Message}' when writing an object");
    }
    catch (Exception e)
    {
        Console.WriteLine($"Unknown encountered on server.
Message: '{e.Message}' when writing an object");
    }
    }
}

```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CompleteMultipartUpload](#)
 - [CreateMultipartUpload](#)
 - [GetObjectMetadata](#)
 - [UploadPartCopy](#)

Transformation de données avec S3 Object Lambda

L'exemple de code suivant illustre comment transformer des données pour votre application avec S3 Object Lambda.

SDK pour .NET

Explique comment ajouter du code personnalisé aux requêtes GET S3 standard afin de modifier l'objet demandé extrait de S3 pour qu'il réponde aux besoins du client ou de l'application qui le demande.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Lambda
- Amazon S3

Charger ou télécharger des fichiers volumineux

L'exemple de code suivant montre comment charger ou télécharger des fichiers volumineux vers et depuis Amazon S3.

Pour plus d'informations, consultez la rubrique [Uploading an object using multipart upload](#) (Chargement d'un objet à l'aide du chargement partitionné).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Appelez des fonctions qui transfèrent des fichiers vers et depuis un compartiment S3 à l'aide d'Amazon S3 TransferUtility.

```
global using System.Text;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.S3.Transfer;
global using TransferUtilityBasics;

// This Amazon S3 client uses the default user credentials
```

```
// defined for this computer.
using Microsoft.Extensions.Configuration;

IAmazonS3 client = new AmazonS3Client();
var transferUtil = new TransferUtility(client);
IConfiguration _configuration;

_configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load test settings from JSON file.
    .AddJsonFile("settings.local.json",
        true) // Optionally load local settings.
    .Build();

// Edit the values in settings.json to use an S3 bucket and files that
// exist on your AWS account and on the local computer where you
// run this scenario.
var bucketName = _configuration["BucketName"];
var localPath =
    $"{Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)}\
\TransferFolder";

DisplayInstructions();

PressEnter();

Console.WriteLine();

// Upload a single file to an S3 bucket.
DisplayTitle("Upload a single file");

var fileToUpload = _configuration["FileToUpload"];
Console.WriteLine($"Uploading {fileToUpload} to the S3 bucket, {bucketName}.");

var success = await TransferMethods.UploadSingleFileAsync(transferUtil, bucketName,
    fileToUpload, localPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the file, {fileToUpload} to
    {bucketName}.");
}

PressEnter();
```

```
// Upload a local directory to an S3 bucket.
DisplayTitle("Upload all files from a local directory");
Console.WriteLine("Upload all the files in a local folder to an S3 bucket.");
const string keyPrefix = "UploadFolder";
var uploadPath = $"{localPath}\\UploadFolder";

Console.WriteLine($"Uploading the files in {uploadPath} to {bucketName}");
DisplayTitle($"{uploadPath} files");
DisplayLocalFiles(uploadPath);
Console.WriteLine();

PressEnter();

success = await TransferMethods.UploadFullDirectoryAsync(transferUtil, bucketName,
    keyPrefix, uploadPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the files in {uploadPath} to
{bucketName}.");
    Console.WriteLine($"{bucketName} currently contains the following files:");
    await DisplayBucketFiles(client, bucketName, keyPrefix);
    Console.WriteLine();
}

PressEnter();

// Download a single file from an S3 bucket.
DisplayTitle("Download a single file");
Console.WriteLine("Now we will download a single file from an S3 bucket.");

var keyName = _configuration["FileToDownload"];

Console.WriteLine($"Downloading {keyName} from {bucketName}.");

success = await TransferMethods.DownloadSingleFileAsync(transferUtil, bucketName,
    keyName, localPath);
if (success)
{
    Console.WriteLine($"Successfully downloaded the file, {keyName} from
{bucketName}.");
}

PressEnter();
```

```
// Download the contents of a directory from an S3 bucket.
DisplayTitle("Download the contents of an S3 bucket");
var s3Path = _configuration["S3Path"];
var downloadPath = $"{localPath}\\{s3Path}";

Console.WriteLine($"Downloading the contents of {bucketName}\\{s3Path}");
Console.WriteLine($"{bucketName}\\{s3Path} contains the following files:");
await DisplayBucketFiles(client, bucketName, s3Path);
Console.WriteLine();

success = await TransferMethods.DownloadS3DirectoryAsync(transferUtil, bucketName,
    s3Path, downloadPath);
if (success)
{
    Console.WriteLine($"Downloaded the files in {bucketName} to {downloadPath}.");
    Console.WriteLine($"{downloadPath} now contains the following files:");
    DisplayLocalFiles(downloadPath);
}

Console.WriteLine("\nThe TransferUtility Basics application has completed.");
PressEnter();

// Displays the title for a section of the scenario.
static void DisplayTitle(string titleText)
{
    var sepBar = new string('-', Console.WindowWidth);

    Console.WriteLine(sepBar);
    Console.WriteLine(CenterText(titleText));
    Console.WriteLine(sepBar);
}

// Displays a description of the actions to be performed by the scenario.
static void DisplayInstructions()
{
    var sepBar = new string('-', Console.WindowWidth);

    DisplayTitle("Amazon S3 Transfer Utility Basics");
    Console.WriteLine("This program shows how to use the Amazon S3 Transfer
Utility.");
    Console.WriteLine("It performs the following actions:");
    Console.WriteLine("\t1. Upload a single object to an S3 bucket.");
    Console.WriteLine("\t2. Upload an entire directory from the local computer to an
\n\t S3 bucket.");
}
```

```

        Console.WriteLine("\t3. Download a single object from an S3 bucket.");
        Console.WriteLine("\t4. Download the objects in an S3 bucket to a local
directory.");
        Console.WriteLine($"\\n{sepBar}");
    }

    // Pauses the scenario.
    static void PressEnter()
    {
        Console.WriteLine("Press <Enter> to continue.");
        _ = Console.ReadLine();
        Console.WriteLine("\\n");
    }

    // Returns the string textToCenter, padded on the left with spaces
    // that center the text on the console display.
    static string CenterText(string textToCenter)
    {
        var centeredText = new StringBuilder();
        var screenWidth = Console.WindowWidth;
        centeredText.Append(new string(' ', (int)(screenWidth - textToCenter.Length) /
2));
        centeredText.Append(textToCenter);
        return centeredText.ToString();
    }

    // Displays a list of file names included in the specified path.
    static void DisplayLocalFiles(string localPath)
    {
        var fileList = Directory.GetFiles(localPath);
        if (fileList.Length > 0)
        {
            foreach (var fileName in fileList)
            {
                Console.WriteLine(fileName);
            }
        }
    }

    // Displays a list of the files in the specified S3 bucket and prefix.
    static async Task DisplayBucketFiles(IAmazonS3 client, string bucketName, string
s3Path)
    {
        ListObjectsV2Request request = new()

```

```

{
    BucketName = bucketName,
    Prefix = s3Path,
    MaxKeys = 5,
};

var response = new ListObjectsV2Response();

do
{
    response = await client.ListObjectsV2Async(request);

    response.S3Objects
        .ForEach(obj => Console.WriteLine($"{obj.Key}"));

    // If the response is truncated, set the request ContinuationToken
    // from the NextContinuationToken property of the response.
    request.ContinuationToken = response.NextContinuationToken;
} while (response.IsTruncated);
}

```

Chargez un seul fichier.

```

/// <summary>
/// Uploads a single file from the local computer to an S3 bucket.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
/// object.</param>
/// <param name="bucketName">The name of the S3 bucket where the file
/// will be stored.</param>
/// <param name="fileName">The name of the file to upload.</param>
/// <param name="localPath">The local path where the file is stored.</param>
/// <returns>A boolean value indicating the success of the action.</returns>
public static async Task<bool> UploadSingleFileAsync(
    TransferUtility transferUtil,
    string bucketName,
    string fileName,
    string localPath)
{
    if (File.Exists($"{localPath}\\{fileName}"))

```

```

        {
            try
            {
                await transferUtil.UploadAsync(new TransferUtilityUploadRequest
                {
                    BucketName = bucketName,
                    Key = fileName,
                    FilePath = $"{localPath}\\{fileName}",
                });

                return true;
            }
            catch (AmazonS3Exception s3Ex)
            {
                Console.WriteLine($"Could not upload {fileName} from {localPath}
because:");
                Console.WriteLine(s3Ex.Message);
                return false;
            }
        }
        else
        {
            Console.WriteLine($"{fileName} does not exist in {localPath}");
            return false;
        }
    }
}

```

Chargez un répertoire local complet.

```

/// <summary>
/// Uploads all the files in a local directory to a directory in an S3
/// bucket.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
/// object.</param>
/// <param name="bucketName">The name of the S3 bucket where the files
/// will be stored.</param>
/// <param name="keyPrefix">The key prefix is the S3 directory where
/// the files will be stored.</param>
/// <param name="localPath">The local directory that contains the files
/// to be uploaded.</param>

```

```

    /// <returns>A Boolean value representing the success of the action.</
returns>
    public static async Task<bool> UploadFullDirectoryAsync(
        TransferUtility transferUtil,
        string bucketName,
        string keyPrefix,
        string localPath)
    {
        if (Directory.Exists(localPath))
        {
            try
            {
                await transferUtil.UploadDirectoryAsync(new
TransferUtilityUploadDirectoryRequest
                {
                    BucketName = bucketName,
                    KeyPrefix = keyPrefix,
                    Directory = localPath,
                });

                return true;
            }
            catch (AmazonS3Exception s3Ex)
            {
                Console.WriteLine($"Can't upload the contents of {localPath}
because:");
                Console.WriteLine(s3Ex?.Message);
                return false;
            }
        }
        else
        {
            Console.WriteLine($"The directory {localPath} does not exist.");
            return false;
        }
    }
}

```

Téléchargez un seul fichier.

```

    /// <summary>

```



```

/// Download a single file from an S3 bucket to the local computer.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
/// object.</param>
/// <param name="bucketName">The name of the S3 bucket containing the
/// file to download.</param>
/// <param name="keyName">The name of the file to download.</param>
/// <param name="localPath">The path on the local computer where the
/// downloaded file will be saved.</param>
/// <returns>A Boolean value indicating the results of the action.</returns>
public static async Task<bool> DownloadSingleFileAsync(
    TransferUtility transferUtil,
        string bucketName,
        string keyName,
        string localPath)
{
    await transferUtil.DownloadAsync(new TransferUtilityDownloadRequest
    {
        BucketName = bucketName,
        Key = keyName,
        FilePath = $"{localPath}\\{keyName}",
    });

    return (File.Exists($"{localPath}\\{keyName}"));
}

```

Téléchargez le contenu d'un compartiment S3.

```

/// <summary>
/// Downloads the contents of a directory in an S3 bucket to a
/// directory on the local computer.
/// </summary>
/// <param name="transferUtil">The transfer initialized TransferUtility
/// object.</param>
param>
/// <param name="bucketName">The bucket containing the files to download.</
param>
/// <param name="s3Path">The S3 directory where the files are located.</
param>
/// <param name="localPath">The local path to which the files will be
/// saved.</param>

```

```
    /// <returns>A Boolean value representing the success of the action.</
returns>
    public static async Task<bool> DownloadS3DirectoryAsync(
        TransferUtility transferUtil,
        string bucketName,
        string s3Path,
        string localPath)
    {
        int fileCount = 0;

        // If the directory doesn't exist, it will be created.
        if (Directory.Exists(s3Path))
        {
            var files = Directory.GetFiles(localPath);
            fileCount = files.Length;
        }

        await transferUtil.DownloadDirectoryAsync(new
TransferUtilityDownloadDirectoryRequest
        {
            BucketName = bucketName,
            LocalDirectory = localPath,
            S3Directory = s3Path,
        });

        if (Directory.Exists(localPath))
        {
            var files = Directory.GetFiles(localPath);
            if (files.Length > fileCount)
            {
                return true;
            }

            // No change in the number of files. Assume
            // the download failed.
            return false;
        }

        // The local directory doesn't exist. No files
        // were downloaded.
        return false;
    }
}
```

Suivez la progression d'un téléchargement à l'aide du TransferUtility.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to track the progress of a multipart upload
/// using the Amazon Simple Storage Service (Amazon S3) TransferUtility to
/// upload to an Amazon S3 bucket.
/// </summary>
public class TrackMPUUsingHighLevelAPI
{
    public static async Task Main()
    {
        string bucketName = "amzn-s3-demo-bucket";
        string keyName = "sample_pic.png";
        string path = "filepath/directory/";
        string filePath = $"{path}{keyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        await TrackMPUAsync(client, bucketName, filePath, keyName);
    }

    /// <summary>
    /// Starts an Amazon S3 multipart upload and assigns an event handler to
    /// track the progress of the upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// perform the multipart upload.</param>
    /// <param name="bucketName">The name of the bucket to which to upload
    /// the file.</param>
    /// <param name="filePath">The path, including the file name of the
    /// file to be uploaded to the Amazon S3 bucket.</param>
    /// <param name="keyName">The file name to be used in the
```

```

    /// destination Amazon S3 bucket.</param>
    public static async Task TrackMPUAsync(
        IAmazonS3 client,
        string bucketName,
        string filePath,
        string keyName)
    {
        try
        {
            var fileTransferUtility = new TransferUtility(client);

            // Use TransferUtilityUploadRequest to configure options.
            // In this example we subscribe to an event.
            var uploadRequest =
                new TransferUtilityUploadRequest
                {
                    BucketName = bucketName,
                    FilePath = filePath,
                    Key = keyName,
                };

            uploadRequest.UploadProgressEvent +=
                new EventHandler<UploadProgressArgs>(
                    UploadRequest_UploadPartProgressEvent);

            await fileTransferUtility.UploadAsync(uploadRequest);
            Console.WriteLine("Upload completed");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error:: {ex.Message}");
        }
    }

    /// <summary>
    /// Event handler to check the progress of the multipart upload.
    /// </summary>
    /// <param name="sender">The object that raised the event.</param>
    /// <param name="e">The object that contains multipart upload
    /// information.</param>
    public static void UploadRequest_UploadPartProgressEvent(object sender,
        UploadProgressArgs e)
    {
        // Process event.
    }

```

```

        Console.WriteLine($"{e.TransferredBytes}/{e.TotalBytes}");
    }
}

```

Chargez un objet avec chiffrement.

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Uses the Amazon Simple Storage Service (Amazon S3) low level API to
/// perform a multipart upload to an Amazon S3 bucket.
/// </summary>
public class SSECLowLevelMPUCopyObject
{
    public static async Task Main()
    {
        string existingBucketName = "amzn-s3-demo-bucket";
        string sourceKeyName = "sample_file.txt";
        string targetKeyName = "sample_file_copy.txt";
        string filePath = $"sample\\{targetKeyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USEast1.
        IAmazonS3 client = new AmazonS3Client();

        // Create the encryption key.
        var base64Key = CreateEncryptionKey();

        await CreateSampleObjUsingClientEncryptionKeyAsync(
            client,
            existingBucketName,
            sourceKeyName,
            filePath,

```

```

        base64Key);
    }

    /// <summary>
    /// Creates the encryption key to use with the multipart upload.
    /// </summary>
    /// <returns>A string containing the base64-encoded key for encrypting
    /// the multipart upload.</returns>
    public static string CreateEncryptionKey()
    {
        Aes aesEncryption = Aes.Create();
        aesEncryption.KeySize = 256;
        aesEncryption.GenerateKey();
        string base64Key = Convert.ToBase64String(aesEncryption.Key);
        return base64Key;
    }

    /// <summary>
    /// Creates and uploads an object using a multipart upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 object used to
    /// initialize and perform the multipart upload.</param>
    /// <param name="existingBucketName">The name of the bucket to which
    /// the object will be uploaded.</param>
    /// <param name="sourceKeyName">The source object name.</param>
    /// <param name="filePath">The location of the source object.</param>
    /// <param name="base64Key">The encryption key to use with the upload.</
param>
    public static async Task CreateSampleObjUsingClientEncryptionKeyAsync(
        IAmazonS3 client,
        string existingBucketName,
        string sourceKeyName,
        string filePath,
        string base64Key)
    {
        List<UploadPartResponse> uploadResponses = new
List<UploadPartResponse>();

        InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = sourceKeyName,

```

```

        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key,
    };

    InitiateMultipartUploadResponse initResponse =
        await client.InitiateMultipartUploadAsync(initiateRequest);

    long contentLength = new FileInfo(filePath).Length;
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

    try
    {
        long filePosition = 0;
        for (int i = 1; filePosition < contentLength; i++)
        {
            UploadPartRequest uploadRequest = new UploadPartRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                UploadId = initResponse.UploadId,
                PartNumber = i,
                PartSize = partSize,
                FilePosition = filePosition,
                FilePath = filePath,
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            // Upload part and add response to our list.
            uploadResponses.Add(await
client.UploadPartAsync(uploadRequest));

            filePosition += partSize;
        }

        CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = sourceKeyName,
            UploadId = initResponse.UploadId,
        };
    };

```

```

        completeRequest.AddPartETags(uploadResponses);

        CompleteMultipartUploadResponse completeUploadResponse =
            await client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (Exception exception)
    {
        Console.WriteLine($"Exception occurred: {exception.Message}");

        // If there was an error, abort the multipart upload.
        AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = sourceKeyName,
            UploadId = initResponse.UploadId,
        };

        await client.AbortMultipartUploadAsync(abortMPURequest);
    }
}
}

```

Exemples sans serveur

Invoker une fonction lambda à partir d'un déclencheur Amazon S3

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par le chargement d'un objet vers un compartiment S3. La fonction extrait le nom du compartiment S3 et la clé de l'objet à partir du paramètre d'événement et appelle l'API Amazon S3 pour récupérer et consigner le type de contenu de l'objet.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Utilisation d'un événement S3 avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");
            }
        }
    }
}
```

```
        var objectResult = await _s3Client.GetObjectAsync(bucket, key);

        context.Logger.LogLine($"Returning {objectResult.Key}");

        return objectResult.Key;
    }
    catch (Exception e)
    {
        context.Logger.LogLine($"Error processing request - {e.Message}");

        return string.Empty;
    }
}
}
```

SageMaker Exemples d'IA utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide de l' SageMaker IA.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Actions](#)
- [Scénarios](#)

Mise en route

Bonjour SageMaker AI

L'exemple de code suivant montre comment commencer à utiliser l' SageMaker IA.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.SageMaker;
using Amazon.SageMaker.Model;

namespace SageMakerActions;

public static class HelloSageMaker
{
    static async Task Main(string[] args)
    {
        var sageMakerClient = new AmazonSageMakerClient();

        Console.WriteLine($"Hello Amazon SageMaker! Let's list some of your notebook
instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five notebook instances.
        var response = await sageMakerClient.ListNotebookInstancesAsync(
            new ListNotebookInstancesRequest()
            {
                MaxResults = 5
            });

        if (!response.NotebookInstances.Any())
        {
            Console.WriteLine($"No notebook instances found.");
        }
    }
}
```

```

        Console.WriteLine("See https://docs.aws.amazon.com/sagemaker/latest/dg/
howitworks-create-ws.html to create one.");
    }

    foreach (var notebookInstance in response.NotebookInstances)
    {
        Console.WriteLine($"Instance:
{notebookInstance.NotebookInstanceName}");
        Console.WriteLine($"Arn: {notebookInstance.NotebookInstanceArn}");
        Console.WriteLine($"Creation Date:
{notebookInstance.CreationTime.ToShortDateString()}");
        Console.WriteLine();
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ListNotebookInstances](#) à la section Référence des AWS SDK pour .NET API.

Actions

CreatePipeline

L'exemple de code suivant montre comment utiliser `CreatePipeline`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create a pipeline from a JSON definition, or update it if the pipeline
already exists.
/// </summary>
/// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)

```

```
{
    try
    {
        var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
            new UpdatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
                PipelineName = name,
                RoleArn = roleArn
            });
        return updateResponse.PipelineArn;
    }
    catch (Amazon.SageMaker.Model.ResourceNotFoundException)
    {
        var createResponse = await _amazonSageMaker.CreatePipelineAsync(
            new CreatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
                PipelineName = name,
                RoleArn = roleArn
            });
        return createResponse.PipelineArn;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CreatePipeline](#) à la section Référence des AWS SDK pour .NET API.

DeletePipeline

L'exemple de code suivant montre comment utiliser `DeletePipeline`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a SageMaker pipeline by name.
/// </summary>
/// <param name="pipelineName">The name of the pipeline to delete.</param>
/// <returns>The ARN of the pipeline.</returns>
public async Task<string> DeletePipelineByName(string pipelineName)
{
    var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
        new DeletePipelineRequest()
        {
            PipelineName = pipelineName
        });

    return deleteResponse.PipelineArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeletePipeline](#) à la section Référence des AWS SDK pour .NET API.

DescribePipelineExecution

L'exemple de code suivant montre comment utiliser `DescribePipelineExecution`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Check the status of a run.
    /// </summary>
    /// <param name="pipelineExecutionArn">The ARN.</param>
    /// <returns>The status of the pipeline.</returns>
    public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
    {
        var describeResponse = await
        _amazonSageMaker.DescribePipelineExecutionAsync(
            new DescribePipelineExecutionRequest()
            {
                PipelineExecutionArn = pipelineExecutionArn
            });

        return describeResponse.PipelineExecutionStatus;
    }

```

- Pour plus de détails sur l'API, reportez-vous [DescribePipelineExecution](#) à la section Référence des AWS SDK pour .NET API.

StartPipelineExecution

L'exemple de code suivant montre comment utiliser `StartPipelineExecution`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Run a pipeline with input and output file locations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
    /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>

```

```

/// <param name="outputLocationUrl">The output location in Amazon S3.</param>
/// <param name="pipelineName">The name of the pipeline.</param>
/// <param name="executionRoleArn">The ARN of the role.</param>
/// <returns>The ARN of the pipeline run.</returns>
public async Task<string> ExecutePipeline(
    string queueUrl,
    string inputLocationUrl,
    string outputLocationUrl,
    string pipelineName,
    string executionRoleArn)
{
    var inputConfig = new VectorEnrichmentJobInputConfig()
    {
        DataSourceConfig = new()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = inputLocationUrl
            }
        },
        DocumentType = VectorEnrichmentJobDocumentType.CSV
    };

    var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
    {
        S3Data = new VectorEnrichmentJobS3Data()
        {
            S3Uri = outputLocationUrl
        }
    };

    var jobConfig = new VectorEnrichmentJobConfig()
    {
        ReverseGeocodingConfig = new ReverseGeocodingConfig()
        {
            XAttributeName = "Longitude",
            YAttributeName = "Latitude"
        }
    };

#pragma warning disable SageMaker1002 // Property value does not match required
    pattern is allowed here to match the pipeline definition.
    var startExecutionResponse = await
        _amazonSageMaker.StartPipelineExecutionAsync(

```



```

        new StartPipelineExecutionRequest()
        {
            PipelineName = pipelineName,
            PipelineExecutionDisplayName = pipelineName + "-example-execution",
            PipelineParameters = new List<Parameter>()
            {
                new Parameter() { Name = "parameter_execution_role", Value =
executionRoleArn },
                new Parameter() { Name = "parameter_queue_url", Value =
queueUrl },
                new Parameter() { Name = "parameter_vej_input_config", Value =
JsonSerializer.Serialize(inputConfig) },
                new Parameter() { Name = "parameter_vej_export_config", Value =
JsonSerializer.Serialize(exportConfig) },
                new Parameter() { Name = "parameter_step_1_vej_config", Value =
JsonSerializer.Serialize(jobConfig) }
            }
        });
#pragma warning restore SageMaker1002
        return startExecutionResponse.PipelineExecutionArn;
    }

```

- Pour plus de détails sur l'API, reportez-vous [StartPipelineExecution](#) à la section Référence des AWS SDK pour .NET API.

UpdatePipeline

L'exemple de code suivant montre comment utiliser `UpdatePipeline`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Create a pipeline from a JSON definition, or update it if the pipeline
already exists.

```

```
/// </summary>
/// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
{
    try
    {
        var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
            new UpdatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
                PipelineName = name,
                RoleArn = roleArn
            });
        return updateResponse.PipelineArn;
    }
    catch (Amazon.SageMaker.Model.ResourceNotFoundException)
    {
        var createResponse = await _amazonSageMaker.CreatePipelineAsync(
            new CreatePipelineRequest()
            {
                PipelineDefinition = pipelineJson,
                PipelineDescription = description,
                PipelineDisplayName = displayName,
                PipelineName = name,
                RoleArn = roleArn
            });
        return createResponse.PipelineArn;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [UpdatePipeline](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Mise en route avec des tâches géospatiales et des pipelines

L'exemple de code suivant illustre comment :

- configurer les ressources d'un pipeline ;
- configurer un pipeline qui exécute une tâche géospatiale ;
- démarrer l'exécution d'un pipeline ;
- surveiller le statut de l'exécution ;
- afficher le résultat du pipeline ;
- nettoyer les ressources.

Pour plus d'informations, consultez [Créer et exécuter des SageMaker pipelines à l'aide AWS SDKs de Community.aws](#).

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une classe qui englobe les opérations d' SageMaker IA.

```
using System.Text.Json;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

namespace SageMakerActions;

/// <summary>
/// Wrapper class for Amazon SageMaker actions and logic.
/// </summary>
public class SageMakerWrapper
```

```

{
    private readonly IAmazonSageMaker _amazonSageMaker;
    public SageMakerWrapper(IAmazonSageMaker amazonSageMaker)
    {
        _amazonSageMaker = amazonSageMaker;
    }

    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
    already exists.
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return createResponse.PipelineArn;
        }
    }
}

```

```

    /// <summary>
    /// Run a pipeline with input and output file locations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
    /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
    /// <param name="outputLocationUrl">The output location in Amazon S3.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="executionRoleArn">The ARN of the role.</param>
    /// <returns>The ARN of the pipeline run.</returns>
    public async Task<string> ExecutePipeline(
        string queueUrl,
        string inputLocationUrl,
        string outputLocationUrl,
        string pipelineName,
        string executionRoleArn)
    {
        var inputConfig = new VectorEnrichmentJobInputConfig()
        {
            DataSourceConfig = new()
            {
                S3Data = new VectorEnrichmentJobS3Data()
                {
                    S3Uri = inputLocationUrl
                }
            },
            DocumentType = VectorEnrichmentJobDocumentType.CSV
        };

        var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = outputLocationUrl
            }
        };

        var jobConfig = new VectorEnrichmentJobConfig()
        {
            ReverseGeocodingConfig = new ReverseGeocodingConfig()
            {
                XAttributeName = "Longitude",
                YAttributeName = "Latitude"
            }
        }
    }

```

```

    }
};

#pragma warning disable SageMaker1002 // Property value does not match required
pattern is allowed here to match the pipeline definition.
    var startExecutionResponse = await
        _amazonSageMaker.StartPipelineExecutionAsync(
            new StartPipelineExecutionRequest()
            {
                PipelineName = pipelineName,
                PipelineExecutionDisplayName = pipelineName + "-example-execution",
                PipelineParameters = new List<Parameter>()
                {
                    new Parameter() { Name = "parameter_execution_role", Value =
executionRoleArn },
                    new Parameter() { Name = "parameter_queue_url", Value =
queueUrl },
                    new Parameter() { Name = "parameter_vej_input_config", Value =
JsonSerializer.Serialize(inputConfig) },
                    new Parameter() { Name = "parameter_vej_export_config", Value =
JsonSerializer.Serialize(exportConfig) },
                    new Parameter() { Name = "parameter_step_1_vej_config", Value =
JsonSerializer.Serialize(jobConfig) }
                }
            });
    return startExecutionResponse.PipelineExecutionArn;
}

/// <summary>
/// Check the status of a run.
/// </summary>
/// <param name="pipelineExecutionArn">The ARN.</param>
/// <returns>The status of the pipeline.</returns>
public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
{
    var describeResponse = await
        _amazonSageMaker.DescribePipelineExecutionAsync(
            new DescribePipelineExecutionRequest()
            {
                PipelineExecutionArn = pipelineExecutionArn
            });
}

```

```

        return describeResponse.PipelineExecutionStatus;
    }

    /// <summary>
    /// Delete a SageMaker pipeline by name.
    /// </summary>
    /// <param name="pipelineName">The name of the pipeline to delete.</param>
    /// <returns>The ARN of the pipeline.</returns>
    public async Task<string> DeletePipelineByName(string pipelineName)
    {
        var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
            new DeletePipelineRequest()
            {
                PipelineName = pipelineName
            });

        return deleteResponse.PipelineArn;
    }
}

```

Créez une fonction qui gère les rappels depuis le pipeline d' SageMaker IA.

```

using System.Text.Json;
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

// Assembly attribute to enable the AWS Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SageMakerLambda;

/// <summary>
/// The AWS Lambda function handler for the Amazon SageMaker pipeline.
/// </summary>
public class SageMakerLambdaFunction

```

```

{
    /// <summary>
    /// Default constructor. This constructor is used by AWS Lambda to construct the
    instance. When invoked in a Lambda environment
    /// the AWS credentials will come from the AWS Identity and Access Management
    (IAM) role associated with the function. The AWS Region will be set to the
    /// Region that the Lambda function is running in.
    /// </summary>
    public SageMakerLambdaFunction()
    {
    }

    /// <summary>
    /// The AWS Lambda function handler that processes events from the SageMaker
    pipeline and starts a job or export.
    /// </summary>
    /// <param name="request">The custom SageMaker pipeline request object.</param>
    /// <param name="context">The Lambda context.</param>
    /// <returns>The dictionary of output parameters.</returns>
    public async Task<Dictionary<string, string>> FunctionHandler(PipelineRequest
    request, ILambdaContext context)
    {
        var geoSpatialClient = new AmazonSageMakerGeospatialClient();
        var sageMakerClient = new AmazonSageMakerClient();
        var responseDictionary = new Dictionary<string, string>();
        context.Logger.LogInformation("Function handler started with request: " +
    JsonSerializer.Serialize(request));
        if (request.Records != null && request.Records.Any())
        {
            context.Logger.LogInformation("Records found, this is a queue event.
    Processing the queue records.");
            foreach (var message in request.Records)
            {
                await ProcessMessageAsync(message, context, geoSpatialClient,
    sageMakerClient);
            }
        }
        else if (!string.IsNullOrEmpty(request.vej_export_config))
        {
            context.Logger.LogInformation("Export configuration found, this is an
    export. Start the Vector Enrichment Job (VEJ) export.");

            var outputConfig =
                JsonSerializer.Deserialize<ExportVectorEnrichmentJobOutputConfig>(

```



```

        request.vej_export_config);

        var exportResponse = await
geoSpatialClient.ExportVectorEnrichmentJobAsync(
    new ExportVectorEnrichmentJobRequest()
    {
        Arn = request.vej_arn,
        ExecutionRoleArn = request.Role,
        OutputConfig = outputConfig
    });
    context.Logger.LogInformation($"Export response:
{JsonSerializer.Serialize(exportResponse)}");
    responseDictionary = new Dictionary<string, string>
    {
        { "export_eoj_status", exportResponse.ExportStatus.ToString() },
        { "vej_arn", exportResponse.Arn }
    };
}
else if (!string.IsNullOrEmpty(request.vej_name))
{
    context.Logger.LogInformation("Vector Enrichment Job name found,
starting the job.");
    var inputConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobInputConfig>(
            request.vej_input_config);

    var jobConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobConfig>(
            request.vej_config);

    var jobResponse = await geoSpatialClient.StartVectorEnrichmentJobAsync(
        new StartVectorEnrichmentJobRequest()
        {
            ExecutionRoleArn = request.Role,
            InputConfig = inputConfig,
            Name = request.vej_name,
            JobConfig = jobConfig
        });
    context.Logger.LogInformation("Job response: " +
JsonSerializer.Serialize(jobResponse));
    responseDictionary = new Dictionary<string, string>
    {
        { "vej_arn", jobResponse.Arn },

```

```

        { "statusCode", jobResponse.HttpStatusCode.ToString() }
    };
}
return responseDictionary;
}

/// <summary>
/// Process a queue message and check the status of a SageMaker job.
/// </summary>
/// <param name="message">The queue message.</param>
/// <param name="context">The Lambda context.</param>
/// <param name="geoClient">The SageMaker GeoSpatial client.</param>
/// <param name="sageMakerClient">The SageMaker client.</param>
/// <returns>Async task.</returns>
private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context,
    AmazonSageMakerGeospatialClient geoClient, AmazonSageMakerClient
sageMakerClient)
{
    context.Logger.LogInformation($"Processed message {message.Body}");

    // Get information about the SageMaker job.
    var payload = JsonSerializer.Deserialize<QueuePayload>(message.Body);
    context.Logger.LogInformation($"Payload token {payload!.token}");
    var token = payload.token;

    if (payload.arguments.ContainsKey("vej_arn"))
    {
        // Use the job ARN and the token to get the job status.
        var job_arn = payload.arguments["vej_arn"];
        context.Logger.LogInformation($"Token: {token}, arn {job_arn}");

        var jobInfo = geoClient.GetVectorEnrichmentJobAsync(
            new GetVectorEnrichmentJobRequest()
            {
                Arn = job_arn
            });
        context.Logger.LogInformation("Job info: " +
JsonSerializer.Serialize(jobInfo));
        if (jobInfo.Result.Status == VectorEnrichmentJobStatus.COMPLETED)
        {
            context.Logger.LogInformation($"Status completed, resuming
pipeline...");
            await sageMakerClient.SendPipelineExecutionStepSuccessAsync(

```

```

        new SendPipelineExecutionStepSuccessRequest()
        {
            CallbackToken = token,
            OutputParameters = new List<OutputParameter>()
            {
                new OutputParameter()
                { Name = "export_status", Value =
jobInfo.Result.Status }
            }
        });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.FAILED)
    {
        context.Logger.LogInformation($"Status failed, stopping
pipeline...");
        await sageMakerClient.SendPipelineExecutionStepFailureAsync(
            new SendPipelineExecutionStepFailureRequest()
            {
                CallbackToken = token,
                FailureReason = jobInfo.Result.ErrorDetails.ErrorMessage
            });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.IN_PROGRESS)
    {
        // Put this message back in the queue to reprocess later.
        context.Logger.LogInformation(
            $"Status still in progress, check back later.");
        throw new("Job still running.");
    }
}
}
}
}
}

```

Exécutez un scénario interactif à une invite de commande.

```

public static class PipelineWorkflow
{
    public static IAMazonIdentityManagementService _iamClient = null!;
    public static SageMakerWrapper _sageMakerWrapper = null!;
    public static IAMazonSQS _sqsClient = null!;
    public static IAMazonS3 _s3Client = null!;
    public static IAMazonLambda _lambdaClient = null!;
}

```

```

public static IConfiguration _configuration = null!;

public static string lambdaFunctionName = "SageMakerExampleFunction";
public static string sageMakerRoleName = "SageMakerExampleRole";
public static string lambdaRoleName = "SageMakerExampleLambdaRole";

private static string[] lambdaRolePolicies = null!;
private static string[] sageMakerRolePolicies = null!;

static async Task Main(string[] args)
{
    var options = new AWSOptions() { Region = RegionEndpoint.USSouth1 };
    // Set up dependency injection for the AWS service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>(options)
                .AddAWSService<IAmazonEC2>(options)
                .AddAWSService<IAmazonSageMaker>(options)
                .AddAWSService<IAmazonSageMakerGeospatial>(options)
                .AddAWSService<IAmazonSQS>(options)
                .AddAWSService<IAmazonS3>(options)
                .AddAWSService<IAmazonLambda>(options)
                .AddTransient<SageMakerWrapper>()
        )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    ServicesSetup(host);
    string queueUrl = "";
    string queueName = _configuration["queueName"];
    string bucketName = _configuration["bucketName"];
    var pipelineName = _configuration["pipelineName"];

```

```

try
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "Welcome to the Amazon SageMaker pipeline example scenario.");
    Console.WriteLine(
        "\nThis example scenario will guide you through setting up and
running an" +
        "\nAmazon SageMaker pipeline. The pipeline uses an AWS Lambda
function and an" +
        "\nAmazon SQS Queue. It runs a vector enrichment reverse geocode job
to" +
        "\nreverse geocode addresses in an input file and store the results
in an export file.");
    Console.WriteLine(new string('-', 80));

    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        "First, we will set up the roles, functions, and queue needed by the
SageMaker pipeline.");
    Console.WriteLine(new string('-', 80));

    var lambdaRoleArn = await CreateLambdaRole();
    var sageMakerRoleArn = await CreateSageMakerRole();
    var functionArn = await SetupLambda(lambdaRoleArn, true);
    queueUrl = await SetupQueue(queueName);
    await SetupBucket(bucketName);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Now we can create and run our pipeline.");
    Console.WriteLine(new string('-', 80));

    await SetupPipeline(sageMakerRoleArn, functionArn, pipelineName);
    var executionArn = await ExecutePipeline(queueUrl, sageMakerRoleArn,
pipelineName, bucketName);
    await WaitForPipelineExecution(executionArn);

    await GetOutputResults(bucketName);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("The pipeline has completed. To view the pipeline and
runs " +
        "in SageMaker Studio, follow these instructions:" +

```

```

        "\nhttps://docs.aws.amazon.com/sagemaker/latest/dg/
pipelines-studio.html");
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Finally, let's clean up our resources.");
        Console.WriteLine(new string('-', 80));

        await CleanupResources(true, queueUrl, pipelineName, bucketName);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("SageMaker pipeline scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources(true, queueUrl, pipelineName, bucketName);
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    _sageMakerWrapper = host.Services.GetRequiredService<SageMakerWrapper>();
    _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    _sqsClient = host.Services.GetRequiredService<IAmazonSQS>();
    _s3Client = host.Services.GetRequiredService<IAmazonS3>();
    _lambdaClient = host.Services.GetRequiredService<IAmazonLambda>();
}

/// <summary>
/// Set up AWS Lambda, either by updating an existing function or creating a new
function.
/// </summary>
/// <param name="roleArn">The role Amazon Resource Name (ARN) to use for the
Lambda function.</param>

```

```

/// <param name="askUser">True to ask the user before updating.</param>
/// <returns>The ARN of the function.</returns>
public static async Task<string> SetupLambda(string roleArn, bool askUser)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Setting up the Lambda function for the pipeline.");
    var handlerName =
"SageMakerLambda::SageMakerLambda.SageMakerLambdaFunction::FunctionHandler";
    var functionArn = "";
    try
    {
        var functionInfo = await _lambdaClient.GetFunctionAsync(new
GetFunctionRequest()
        {
            FunctionName = lambdaFunctionName
        });

        var updateFunction = true;
        if (askUser)
        {
            updateFunction = GetYesNoResponse(
                $"\\tThe Lambda function {lambdaFunctionName} already exists, do
you want to update it?");
        }

        if (updateFunction)
        {
            // Update the Lambda function.
            using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
            await _lambdaClient.UpdateFunctionCodeAsync(
                new UpdateFunctionCodeRequest()
                {
                    FunctionName = lambdaFunctionName,
                    ZipFile = zipMemoryStream,
                });
        }

        functionArn = functionInfo.Configuration.FunctionArn;
    }
    catch (ResourceNotFoundException)
    {
        Console.WriteLine($"\\tThe Lambda function {lambdaFunctionName} was not
found, creating the new function.");
    }
}

```

```

        // Create the function if it does not already exist.
        using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
        var createResult = await _lambdaClient.CreateFunctionAsync(
            new CreateFunctionRequest()
            {
                FunctionName = lambdaFunctionName,
                Runtime = Runtime.Dotnet6,
                Description = "SageMaker example function.",
                Code = new FunctionCode()
                {
                    ZipFile = zipMemoryStream
                },
                Handler = handlerName,
                Role = roleArn,
                Timeout = 30
            });

        functionArn = createResult.FunctionArn;
    }

    Console.WriteLine($"\\tLambda ready with ARN {functionArn}.");
    Console.WriteLine(new string('-', 80));
    return functionArn;
}

/// <summary>
/// Create a role to be used by AWS Lambda. Does not create the role if it
already exists.
/// </summary>
/// <returns>The role ARN.</returns>
public static async Task<string> CreateLambdaRole()
{
    Console.WriteLine(new string('-', 80));

    lambdaRolePolicies = new string[]{
        "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
        "arn:aws:iam::aws:policy/AmazonSQSFullAccess",
        "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerGeospatialFullAccess",
        "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy",
    };
}

```



```

        "arn:aws:iam::aws:policy/service-role/" +
        "AWSLambdaSQSQueueExecutionRole"
    };

    var roleArn = await GetRoleArnIfExists(lambdaRoleName);
    if (!string.IsNullOrEmpty(roleArn))
    {
        return roleArn;
    }

    Console.WriteLine("\tCreating a role to for AWS Lambda to use.");

    var assumeRolePolicy = "{" +
        "\"Version\": \"2012-10-17\"," +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\"," +
            "\"Principal\": {" +
                "\"Service\": [" +
                    "\"sagemaker.amazonaws.com\"," +
                    "\"sagemaker-geospatial.amazonaws.com"
\"," +
                    "\"lambda.amazonaws.com\"," +
                    "\"s3.amazonaws.com\"" +
                "]" +
            }," +
            "\"Action\": \"sts:AssumeRole\"" +
        "]" +
    "}";

    var roleResult = await _iamClient!.CreateRoleAsync(
        new CreateRoleRequest()
        {
            AssumeRolePolicyDocument = assumeRolePolicy,
            Path = "/",
            RoleName = lambdaRoleName
        });
    foreach (var policy in lambdaRolePolicies)
    {
        await _iamClient.AttachRolePolicyAsync(
            new AttachRolePolicyRequest()
            {
                PolicyArn = policy,
                RoleName = lambdaRoleName
            });
    }

```

```

    }

    // Allow time for the role to be ready.
    Thread.Sleep(10000);
    Console.WriteLine($"\\tRole ready with ARN {roleResult.Role.Arn}.");
    Console.WriteLine(new string('-', 80));

    return roleResult.Role.Arn;
}

/// <summary>
/// Create a role to be used by SageMaker.
/// </summary>
/// <returns>The role Amazon Resource Name (ARN).</returns>
public static async Task<string> CreateSageMakerRole()
{
    Console.WriteLine(new string('-', 80));

    sageMakerRolePolicies = new string[]{
        "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
        "arn:aws:iam::aws:policy/AmazonSageMakerGeospatialFullAccess",
    };

    var roleArn = await GetRoleArnIfExists(sageMakerRoleName);
    if (!string.IsNullOrEmpty(roleArn))
    {
        return roleArn;
    }

    Console.WriteLine("\\tCreating a role to use with SageMaker.");

    var assumeRolePolicy = "{" +
        "\\\"Version\\\": \"2012-10-17\\\", \" +
        "\\\"Statement\\\": [{\" +
            "\\\"Effect\\\": \"Allow\\\", \" +
            "\\\"Principal\\\": {\" +
                $\"\\\"Service\\\": [\" +
                    \"\\\"sagemaker.amazonaws.com\\\", \" +
                    \"\\\"sagemaker-geospatial.amazonaws.com\\\", \" +
                    \"\\\"lambda.amazonaws.com\\\", \" +
                    \"\\\"s3.amazonaws.com\\\"\" +
                "]" +
            "}" +
    };

```

```

        "}," +
        "\"Action\": \"sts:AssumeRole\"\" +
        "}]\" +
        "\"";

var roleResult = await _iamClient!.CreateRoleAsync(
    new CreateRoleRequest()
    {
        AssumeRolePolicyDocument = assumeRolePolicy,
        Path = "/",
        RoleName = sageMakerRoleName
    });

foreach (var policy in sageMakerRolePolicies)
{
    await _iamClient.AttachRolePolicyAsync(
        new AttachRolePolicyRequest()
        {
            PolicyArn = policy,
            RoleName = sageMakerRoleName
        });
}

// Allow time for the role to be ready.
Thread.Sleep(10000);
Console.WriteLine($"\\tRole ready with ARN {roleResult.Role.Arn}.");
Console.WriteLine(new string('-', 80));
return roleResult.Role.Arn;
}

/// <summary>
/// Set up the SQS queue to use with the pipeline.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <returns>The URL for the queue.</returns>
public static async Task<string> SetupQueue(string queueName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up queue {queueName}.");

    try
    {
        var queueInfo = await _sqsClient.GetQueueUrlAsync(new
GetQueueUrlRequest()

```

```

        { QueueName = queueName });
        return queueInfo.QueueUrl;
    }
    catch (QueueDoesNotExistException)
    {
        var attrs = new Dictionary<string, string>
        {
            {
                QueueAttributeName.DelaySeconds,
                "5"
            },
            {
                QueueAttributeName.ReceiveMessageWaitTimeSeconds,
                "5"
            },
            {
                QueueAttributeName.VisibilityTimeout,
                "300"
            },
        };

        var request = new CreateQueueRequest
        {
            Attributes = attrs,
            QueueName = queueName,
        };

        var response = await _sqsClient.CreateQueueAsync(request);
        Thread.Sleep(10000);
        await ConnectLambda(response.QueueUrl);
        Console.WriteLine($"\\tQueue ready with Url {response.QueueUrl}.");
        Console.WriteLine(new string('-', 80));
        return response.QueueUrl;
    }
}

/// <summary>
/// Connect the queue to the Lambda function as an event source.
/// </summary>
/// <param name="queueUrl">The URL for the queue.</param>
/// <returns>Async task.</returns>
public static async Task ConnectLambda(string queueUrl)
{
    Console.WriteLine(new string('-', 80));

```

```

        Console.WriteLine($"Connecting the Lambda function and queue for the
pipeline.");

        var queueAttributes = await _sqsClient.GetQueueAttributesAsync(
            new GetQueueAttributesRequest() { QueueUrl = queueUrl, AttributeNames =
new List<string>() { "All" } });
        var queueArn = queueAttributes.QueueARN;

        var eventSource = await _lambdaClient.ListEventSourceMappingsAsync(
            new ListEventSourceMappingsRequest()
            {
                FunctionName = lambdaFunctionName
            });

        if (!eventSource.EventSourceMappings.Any())
        {
            // Only add the event source mapping if it does not already exist.
            await _lambdaClient.CreateEventSourceMappingAsync(
                new CreateEventSourceMappingRequest()
                {
                    EventSourceArn = queueArn,
                    FunctionName = lambdaFunctionName,
                    Enabled = true
                });
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the bucket to use for pipeline input and output.
    /// </summary>
    /// <param name="bucketName">The name for the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task SetupBucket(string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up bucket {bucketName}.");

        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            bucketName);

        if (!bucketExists)

```

```

    {
        await _s3Client.PutBucketAsync(new PutBucketRequest()
        {
            BucketName = bucketName,
            BucketRegion = S3Region.USWest2
        });

        Thread.Sleep(5000);

        await _s3Client.PutObjectAsync(new PutObjectRequest()
        {
            BucketName = bucketName,
            Key = "samplefiles/latlongtest.csv",
            FilePath = "latlongtest.csv"
        });
    }

    Console.WriteLine($"\\tBucket {bucketName} ready.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Display some results from the output directory.
/// </summary>
/// <param name="bucketName">The name for the bucket.</param>
/// <returns>Async task.</returns>
public static async Task<string> GetOutputResults(string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Getting output results {bucketName}.");
    string outputKey = "";
    Thread.Sleep(15000);
    var outputFiles = await _s3Client.ListObjectsAsync(
        new ListObjectsRequest()
        {
            BucketName = bucketName,
            Prefix = "outputfiles/"
        });

    if (outputFiles.S3Objects.Any())
    {
        var sampleOutput = outputFiles.S3Objects.OrderBy(s =>
s.LastModified).Last();
        Console.WriteLine($"\\tOutput file: {sampleOutput.Key}");
    }
}

```

```

        var outputSampleResponse = await _s3Client.GetObjectAsync(
            new GetObjectRequest()
            {
                BucketName = bucketName,
                Key = sampleOutput.Key
            });
        outputKey = sampleOutput.Key;
        StreamReader reader = new
StreamReader(outputSampleResponse.ResponseStream);
        await reader.ReadLineAsync();
        Console.WriteLine("\tOutput file contents: \n");
        for (int i = 0; i < 10; i++)
        {
            if (!reader.EndOfStream)
            {
                Console.WriteLine("\t" + await reader.ReadLineAsync());
            }
        }
    }

    Console.WriteLine(new string('-', 80));
    return outputKey;
}

/// <summary>
/// Create a pipeline from the example pipeline JSON
/// that includes the Lambda, callback, processing, and export jobs.
/// </summary>
/// <param name="roleArn">The ARN of the role for the pipeline.</param>
/// <param name="functionArn">The ARN of the Lambda function for the pipeline.</
param>
/// <param name="pipelineName">The name for the pipeline.</param>
/// <returns>The ARN of the pipeline.</returns>
    public static async Task<string> SetupPipeline(string roleArn, string
functionArn, string pipelineName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up the pipeline.");

        var pipelineJson = await File.ReadAllTextAsync("GeoSpatialPipeline.json");

        // Add the correct function ARN instead of the placeholder.
        pipelineJson = pipelineJson.Replace("*FUNCTION_ARN*", functionArn);

```

```

        var pipelineArn = await _sageMakerWrapper.SetupPipeline(pipelineJson,
            roleArn, pipelineName,
            "sdk example pipeline", pipelineName);

        Console.WriteLine($"\\tPipeline set up with ARN {pipelineArn}.");
        Console.WriteLine(new string('-', 80));

        return pipelineArn;
    }

    /// <summary>
    /// Start a pipeline run with job configurations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue used in the pipeline.</param>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>The pipeline run ARN.</returns>
    public static async Task<string> ExecutePipeline(
        string queueUrl,
        string roleArn,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Starting pipeline execution.");

        var input = $"s3://{bucketName}/samplefiles/latlongtest.csv";
        var output = $"s3://{bucketName}/outputfiles/";

        var executionARN =
            await _sageMakerWrapper.ExecutePipeline(queueUrl, input, output,
                pipelineName, roleArn);

        Console.WriteLine($"\\tRun started with ARN {executionARN}.");
        Console.WriteLine(new string('-', 80));

        return executionARN;
    }

    /// <summary>
    /// Wait for a pipeline run to complete.
    /// </summary>
    /// <param name="executionArn">The pipeline run ARN.</param>

```



```

    /// <returns>Async task.</returns>
    public static async Task WaitForPipelineExecution(string executionArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Waiting for pipeline to finish.");

        PipelineExecutionStatus status;
        do
        {
            status = await
_sageMakerWrapper.CheckPipelineExecutionStatus(executionArn);
            Thread.Sleep(30000);
            Console.WriteLine($"\\tStatus is {status}.");
        } while (status == PipelineExecutionStatus.Executing);

        Console.WriteLine($"\\tPipeline finished with status {status}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="askUser">True to ask the user for cleanup.</param>
    /// <param name="queueUrl">The URL of the queue to clean up.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> CleanupResources(
        bool askUser,
        string queueUrl,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (!askUser || GetYesNoResponse($"\\tDelete pipeline {pipelineName}? (y/
n)"))
        {
            Console.WriteLine($"\\tDeleting pipeline.");
            // Delete the pipeline.
            await _sageMakerWrapper.DeletePipelineByName(pipelineName);
        }
    }

```

```

        if (!string.IsNullOrEmpty(queueUrl) && (!askUser ||
GetYesNoResponse($"\\tDelete queue {queueUrl}? (y/n)")))
        {
            Console.WriteLine($"\\tDeleting queue.");
            // Delete the queue.
            await _sqsClient.DeleteQueueAsync(new DeleteQueueRequest(queueUrl));
        }

        if (!askUser || GetYesNoResponse($"\\tDelete Amazon S3 bucket {bucketName}?
(y/n)"))
        {
            Console.WriteLine($"\\tDeleting bucket.");
            // Delete all objects in the bucket.
            var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            if (deleteList.KeyCount > 0)
            {
                await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.S3Objects
                        .Select(o => new KeyVersion { Key = o.Key }).ToList()
                });
            }

            // Now delete the bucket.
            await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
            {
                BucketName = bucketName
            });
        }

        if (!askUser || GetYesNoResponse($"\\tDelete lambda {lambdaFunctionName}? (y/
n)"))
        {
            Console.WriteLine($"\\tDeleting lambda function.");

            await _lambdaClient.DeleteFunctionAsync(new DeleteFunctionRequest()
            {
                FunctionName = lambdaFunctionName
            });
        }

```

```

    }

    if (!askUser || GetYesNoResponse($"\tDelete role {lambdaRoleName}? (y/n)"))
    {
        Console.WriteLine($" \tDetaching policies and deleting role.");

        foreach (var policy in lambdaRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = lambdaRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = lambdaRoleName
        });
    }

    if (!askUser || GetYesNoResponse($" \tDelete role {sageMakerRoleName}? (y/
n)"))
    {
        Console.WriteLine($" \tDetaching policies and deleting role.");

        foreach (var policy in sageMakerRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = sageMakerRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = sageMakerRoleName
        });
    }

    Console.WriteLine(new string('-', 80));

```

```

        return true;
    }

    /// <summary>
    /// Helper method to get a role's ARN if it already exists.
    /// </summary>
    /// <param name="roleName">The name of the AWS Identity and Access Management
    (IAM) Role to look for.</param>
    /// <returns>The role ARN if it exists, otherwise an empty string.</returns>
    private static async Task<string> GetRoleArnIfExists(string roleName)
    {
        Console.WriteLine($"Checking for role named {roleName}.");

        try
        {
            var existingRole = await _iamClient.GetRoleAsync(new GetRoleRequest()
            {
                RoleName = roleName
            });
            return existingRole.Role.Arn;
        }
        catch (NoSuchEntityException)
        {
            return string.Empty;
        }
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}

```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Exemples d'utilisation de Secrets Manager SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with Secrets Manager.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

GetSecretValue

L'exemple de code suivant montre comment utiliser `GetSecretValue`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.SecretsManager;
using Amazon.SecretsManager.Model;

/// <summary>
/// This example uses the Amazon Web Service Secrets Manager to retrieve
/// the secret value for the provided secret name.
/// </summary>
public class GetSecretValue
{
    /// <summary>
    /// The main method initializes the necessary values and then calls
    /// the GetSecretAsync and DecodeString methods to get the decoded
    /// secret value for the secret named in secretName.
    /// </summary>
    public static async Task Main()
    {
        string secretName = "<{{MySecretName}}>";
        string secret;

        IAmazonSecretsManager client = new AmazonSecretsManagerClient();

        var response = await GetSecretAsync(client, secretName);

        if (response is not null)
        {
            secret = DecodeString(response);

            if (!string.IsNullOrEmpty(secret))
            {
                Console.WriteLine($"The decoded secret value is: {secret}.");
            }
            else
            {
                Console.WriteLine("No secret value was returned.");
            }
        }
    }

    /// <summary>
    /// Retrieves the secret value given the name of the secret to
```

```

    /// retrieve.
    /// </summary>
    /// <param name="client">The client object used to retrieve the secret
    /// value for the given secret name.</param>
    /// <param name="secretName">The name of the secret value to retrieve.</
param>
    /// <returns>The GetSecretValueReponse object returned by
    /// GetSecretValueAsync.</returns>
    public static async Task<GetSecretValueResponse> GetSecretAsync(
        IAmazonSecretsManager client,
        string secretName)
    {
        GetSecretValueRequest request = new GetSecretValueRequest()
        {
            SecretId = secretName,
            VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT
if unspecified.
        };

        GetSecretValueResponse response = null;

        // For the sake of simplicity, this example handles only the most
        // general SecretsManager exception.
        try
        {
            response = await client.GetSecretValueAsync(request);
        }
        catch (AmazonSecretsManagerException e)
        {
            Console.WriteLine($"Error: {e.Message}");
        }

        return response;
    }

    /// <summary>
    /// Decodes the secret returned by the call to GetSecretValueAsync and
    /// returns it to the calling program.
    /// </summary>
    /// <param name="response">A GetSecretValueResponse object containing
    /// the requested secret value returned by GetSecretValueAsync.</param>
    /// <returns>A string representing the decoded secret value.</returns>
    public static string DecodeString(GetSecretValueResponse response)
    {

```

```
// Decrypts secret using the associated AWS Key Management Service
// Customer Master Key (CMK.) Depending on whether the secret is a
// string or binary value, one of these fields will be populated.
if (response.SecretString is not null)
{
    var secret = response.SecretString;
    return secret;
}
else if (response.SecretBinary is not null)
{
    var memoryStream = response.SecretBinary;
    StreamReader reader = new StreamReader(memoryStream);
    string decodedBinarySecret =
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(reader.ReadToEnd()));
    return decodedBinarySecret;
}
else
{
    return string.Empty;
}
}
```

- Pour plus de détails sur l'API, reportez-vous [GetSecretValue](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon SES utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon SES.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

CreateTemplate

L'exemple de code suivant montre comment utiliser `CreateTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create an email template.
/// </summary>
/// <param name="name">Name of the template.</param>
/// <param name="subject">Email subject.</param>
/// <param name="text">Email body text.</param>
/// <param name="html">Email HTML body text.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateEmailTemplateAsync(string name, string subject,
string text,
    string html)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.CreateTemplateAsync(
            new CreateTemplateRequest
            {
```

```
        Template = new Template
        {
            TemplateName = name,
            SubjectPart = subject,
            TextPart = text,
            HtmlPart = html
        }
    });
    success = response.HttpStatusCode == HttpStatusCode.OK;
}
catch (Exception ex)
{
    Console.WriteLine("CreateEmailTemplateAsync failed with exception: " +
ex.Message);
}

return success;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateTemplate](#) à la section Référence des AWS SDK pour .NET API.

DeleteIdentity

L'exemple de code suivant montre comment utiliser `DeleteIdentity`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an email identity.
/// </summary>
/// <param name="identityEmail">The identity email to delete.</param>
```

```
/// <returns>True if successful.</returns>
public async Task<bool> DeleteIdentityAsync(string identityEmail)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.DeleteIdentityAsync(
            new DeleteIdentityRequest
            {
                Identity = identityEmail
            });
        success = response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Exception ex)
    {
        Console.WriteLine("DeleteIdentityAsync failed with exception: " +
            ex.Message);
    }

    return success;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteIdentity](#) à la section Référence des AWS SDK pour .NET API.

DeleteTemplate

L'exemple de code suivant montre comment utiliser `DeleteTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
```

```
/// Delete an email template.
/// </summary>
/// <param name="templateName">Name of the template.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailTemplateAsync(string templateName)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.DeleteTemplateAsync(
            new DeleteTemplateRequest
            {
                TemplateName = templateName
            });
        success = response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Exception ex)
    {
        Console.WriteLine("DeleteEmailTemplateAsync failed with exception: " +
            ex.Message);
    }

    return success;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteTemplate](#) à la section Référence des AWS SDK pour .NET API.

GetIdentityVerificationAttributes

L'exemple de code suivant montre comment utiliser `GetIdentityVerificationAttributes`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get identity verification status for an email.
/// </summary>
/// <returns>The verification status of the email.</returns>
public async Task<VerificationStatus> GetIdentityStatusAsync(string email)
{
    var result = VerificationStatus.TemporaryFailure;
    try
    {
        var response =
            await
                _amazonSimpleEmailService.GetIdentityVerificationAttributesAsync(
                    new GetIdentityVerificationAttributesRequest
                    {
                        Identities = new List<string> { email }
                    });

        if (response.VerificationAttributes.ContainsKey(email))
            result = response.VerificationAttributes[email].VerificationStatus;
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetIdentityStatusAsync failed with exception: " +
            ex.Message);
    }

    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetIdentityVerificationAttributes](#) à la section Référence des AWS SDK pour .NET API.

GetSendQuota

L'exemple de code suivant montre comment utiliser `GetSendQuota`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information on the current account's send quota.
/// </summary>
/// <returns>The send quota response data.</returns>
public async Task<GetSendQuotaResponse> GetSendQuotaAsync()
{
    var result = new GetSendQuotaResponse();
    try
    {
        var response = await _amazonSimpleEmailService.GetSendQuotaAsync(
            new GetSendQuotaRequest());
        result = response;
    }
    catch (Exception ex)
    {
        Console.WriteLine("GetSendQuotaAsync failed with exception: " +
ex.Message);
    }

    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetSendQuota](#) à la section Référence des AWS SDK pour .NET API.

ListIdentities

L'exemple de code suivant montre comment utiliser `ListIdentities`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the identities of a specified type for the current account.
/// </summary>
/// <param name="identityType">IdentityType to list.</param>
/// <returns>The list of identities.</returns>
public async Task<List<string>> ListIdentitiesAsync(IdentityType identityType)
{
    var result = new List<string>();
    try
    {
        var response = await _amazonSimpleEmailService.ListIdentitiesAsync(
            new ListIdentitiesRequest
            {
                IdentityType = identityType
            });
        result = response.Identities;
    }
    catch (Exception ex)
    {
        Console.WriteLine("ListIdentitiesAsync failed with exception: " +
ex.Message);
    }

    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListIdentities](#) à la section Référence des AWS SDK pour .NET API.

ListTemplates

L'exemple de code suivant montre comment utiliser `ListTemplates`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// List email templates for the current account.
/// </summary>
/// <returns>A list of template metadata.</returns>
public async Task<List<TemplateMetadata>> ListEmailTemplatesAsync()
{
    var result = new List<TemplateMetadata>();
    try
    {
        var response = await _amazonSimpleEmailService.ListTemplatesAsync(
            new ListTemplatesRequest());
        result = response.TemplatesMetadata;
    }
    catch (Exception ex)
    {
        Console.WriteLine("ListEmailTemplatesAsync failed with exception: " +
ex.Message);
    }

    return result;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListTemplates](#) à la section Référence des AWS SDK pour .NET API.

SendEmail

L'exemple de code suivant montre comment utiliser `SendEmail`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
///  Send an email by using Amazon SES.
/// </summary>
/// <param name="toAddresses">List of recipients.</param>
/// <param name="ccAddresses">List of cc recipients.</param>
/// <param name="bccAddresses">List of bcc recipients.</param>
/// <param name="bodyHtml">Body of the email in HTML.</param>
/// <param name="bodyText">Body of the email in plain text.</param>
/// <param name="subject">Subject line of the email.</param>
/// <param name="senderAddress">From address.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendEmailAsync(List<string> toAddresses,
    List<string> ccAddresses, List<string> bccAddresses,
    string bodyHtml, string bodyText, string subject, string senderAddress)
{
    var messageId = "";
    try
    {
        var response = await _amazonSimpleEmailService.SendEmailAsync(
            new SendEmailRequest
            {
                Destination = new Destination
                {
                    BccAddresses = bccAddresses,
                    CcAddresses = ccAddresses,
                    ToAddresses = toAddresses
                },
                Message = new Message
                {
                    Body = new Body
```

```
        {
            Html = new Content
            {
                Charset = "UTF-8",
                Data = bodyHtml
            },
            Text = new Content
            {
                Charset = "UTF-8",
                Data = bodyText
            }
        },
        Subject = new Content
        {
            Charset = "UTF-8",
            Data = subject
        }
    },
    Source = senderAddress
));
messageId = response.MessageId;
}
catch (Exception ex)
{
    Console.WriteLine("SendEmailAsync failed with exception: " +
ex.Message);
}

return messageId;
}
```

- Pour plus de détails sur l'API, reportez-vous [SendEmail](#) à la section Référence des AWS SDK pour .NET API.

SendTemplatedEmail

L'exemple de code suivant montre comment utiliser `SendTemplatedEmail`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Send an email using a template.
/// </summary>
/// <param name="sender">Address of the sender.</param>
/// <param name="recipients">Addresses of the recipients.</param>
/// <param name="templateName">Name of the email template.</param>
/// <param name="templateDataObject">Data for the email template.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendTemplateEmailAsync(string sender, List<string>
recipients,
    string templateName, object templateDataObject)
{
    var messageId = "";
    try
    {
        // Template data should be serialized JSON from either a class or a
dynamic object.
        var templateData = JsonSerializer.Serialize(templateDataObject);

        var response = await _amazonSimpleEmailService.SendTemplatedEmailAsync(
            new SendTemplatedEmailRequest
            {
                Source = sender,
                Destination = new Destination
                {
                    ToAddresses = recipients
                },
                Template = templateName,
                TemplateData = templateData
            });
        messageId = response.MessageId;
    }
    catch (Exception ex)
```

```

    {
        Console.WriteLine("SendTemplateEmailAsync failed with exception: " +
            ex.Message);
    }

    return messageId;
}

```

- Pour plus de détails sur l'API, reportez-vous [SendTemplatedEmail](#) à la section Référence des AWS SDK pour .NET API.

VerifyEmailIdentity

L'exemple de code suivant montre comment utiliser `VerifyEmailIdentity`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Starts verification of an email identity. This request sends an email
/// from Amazon SES to the specified email address. To complete
/// verification, follow the instructions in the email.
/// </summary>
/// <param name="recipientEmailAddress">Email address to verify.</param>
/// <returns>True if successful.</returns>
public async Task<bool> VerifyEmailIdentityAsync(string recipientEmailAddress)
{
    var success = false;
    try
    {
        var response = await _amazonSimpleEmailService.VerifyEmailIdentityAsync(
            new VerifyEmailIdentityRequest
            {

```

```
        EmailAddress = recipientEmailAddress
    });

    success = response.HttpStatusCode == HttpStatusCode.OK;
}
catch (Exception ex)
{
    Console.WriteLine("VerifyEmailIdentityAsync failed with exception: " +
ex.Message);
}

return success;
}
```

- Pour plus de détails sur l'API, reportez-vous [VerifyEmailIdentity](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Créer une application web pour suivre les données DynamoDB

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une table Amazon DynamoDB et envoie des rapports à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser l'API Amazon DynamoDB .NET pour créer une application web dynamique qui suit les données de travail DynamoDB.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- DynamoDB
- Amazon SES

Créer un outil de suivi des éléments de travail sans serveur Aurora

L'exemple de code suivant montre comment créer une application Web qui suit des éléments de travail dans une base de données Amazon Aurora sans serveur et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES).

SDK pour .NET

Montre comment utiliser le AWS SDK pour .NET pour créer une application Web qui suit les éléments de travail dans une base de données Amazon Aurora et envoie des rapports par e-mail à l'aide d'Amazon Simple Email Service (Amazon SES). Cet exemple utilise un front-end créé avec React.js pour interagir avec un backend RESTful .NET.

- Intégrez une application Web React à AWS des services.
- Listez, ajoutez et mettez à jour des éléments dans une table Aurora.
- Envoyez un rapport par e-mail sur les éléments de travail filtrés à l'aide d'Amazon SES.
- Déployez et gérez des exemples de ressources à l'aide du AWS CloudFormation script inclus.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Aurora
- Amazon RDS
- Services de données Amazon RDS
- Amazon SES

Détecter des objets dans des images

L'exemple de code suivant montre comment créer une application qui utilise Amazon Rekognition afin de détecter des objets par catégorie dans des images.

SDK pour .NET

Montre comment utiliser l'API Java Amazon Rekognition afin de créer une application qui, avec Amazon Rekognition, permet d'identifier des objets par catégorie dans des images stockées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à

l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemples d'API Amazon SES v2 utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de l' AWS SDK pour .NET API v2 d'Amazon SES.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

CreateContact

L'exemple de code suivant montre comment utiliser `CreateContact`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Creates a contact and adds it to the specified contact list.
/// </summary>
/// <param name="emailAddress">The email address of the contact.</param>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The response from the CreateContact operation.</returns>
public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
{
    var request = new CreateContactRequest
    {
        EmailAddress = emailAddress,
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
        Console.WriteLine(ex.Message);
        return true;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
```



```
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateContact](#) à la section Référence des AWS SDK pour .NET API.

CreateContactList

L'exemple de code suivant montre comment utiliser `CreateContactList`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Creates a contact list with the specified name.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateContactListAsync(string contactListName)
{
    var request = new CreateContactListRequest
    {
        ContactListName = contactListName
    };
};
```

```
try
{
    var response = await _sesClient.CreateContactListAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
catch (AlreadyExistsException ex)
{
    Console.WriteLine($"Contact list with name {contactListName} already
exists.");
    Console.WriteLine(ex.Message);
    return true;
}
catch (LimitExceededException ex)
{
    Console.WriteLine("The limit for contact lists has been exceeded.");
    Console.WriteLine(ex.Message);
}
catch (TooManyRequestsException ex)
{
    Console.WriteLine("Too many requests were made. Please try again
later.");
    Console.WriteLine(ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
}
return false;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateContactList](#) à la section Référence des AWS SDK pour .NET API.

CreateEmailIdentity

L'exemple de code suivant montre comment utiliser `CreateEmailIdentity`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Creates an email identity (email address or domain) and starts the
verification process.
/// </summary>
/// <param name="emailIdentity">The email address or domain to create and
verify.</param>
/// <returns>The response from the CreateEmailIdentity operation.</returns>
public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
{
    var request = new CreateEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.CreateEmailIdentityAsync(request);
        return response;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email identity {emailIdentity} already exists.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (LimitExceededException ex)
```

```
{
    Console.WriteLine("The limit for email identities has been exceeded.");
    Console.WriteLine(ex.Message);
    throw;
}
catch (NotFoundException ex)
{
    Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
    Console.WriteLine(ex.Message);
    throw;
}
catch (TooManyRequestsException ex)
{
    Console.WriteLine("Too many requests were made. Please try again
later.");
    Console.WriteLine(ex.Message);
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
    throw;
}
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateEmailIdentity](#) à la section Référence des AWS SDK pour .NET API.

CreateEmailTemplate

L'exemple de code suivant montre comment utiliser `CreateEmailTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
    }

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email template:
{ex.Message}");
    }

    return false;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateEmailTemplate](#) à la section Référence des AWS SDK pour .NET API.

DeleteContactList

L'exemple de code suivant montre comment utiliser `DeleteContactList`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Deletes a contact list and all contacts within it.
/// </summary>
/// <param name="contactListName">The name of the contact list to delete.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteContactListAsync(string contactListName)
{
    var request = new DeleteContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {

```

```
        var response = await _sesClient.DeleteContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
    }

    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteContactList](#) à la section Référence des AWS SDK pour .NET API.

DeleteEmailIdentity

L'exemple de code suivant montre comment utiliser `DeleteEmailIdentity`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Deletes an email identity (email address or domain).
/// </summary>
/// <param name="emailIdentity">The email address or domain to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
{
    var request = new DeleteEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.DeleteEmailIdentityAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
}
```



```
    }  
    catch (Exception ex)  
    {  
        Console.WriteLine($"An error occurred while deleting the email identity:  
{ex.Message}");  
    }  
  
    return false;  
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteEmailIdentity](#) à la section Référence des AWS SDK pour .NET API.

DeleteEmailTemplate

L'exemple de code suivant montre comment utiliser `DeleteEmailTemplate`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Deletes an email template.  
/// </summary>  
/// <param name="templateName">The name of the email template to delete.</param>  
/// <returns>True if successful.</returns>  
public async Task<bool> DeleteEmailTemplateAsync(string templateName)  
{  
    var request = new DeleteEmailTemplateRequest  
    {  
        TemplateName = templateName  
    };  
  
    try  
    {  
        var response = await _sesClient.DeleteEmailTemplateAsync(request);  
    }  
}
```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email template {templateName} does not exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
    }

    return false;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteEmailTemplate](#) à la section Référence des AWS SDK pour .NET API.

ListContacts

L'exemple de code suivant montre comment utiliser `ListContacts`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Lists the contacts in the specified contact list.
/// </summary>
```

```

    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
        }

        return new List<Contact>();
    }

```

- Pour plus de détails sur l'API, reportez-vous [ListContacts](#) à la section Référence des AWS SDK pour .NET API.

SendEmail

L'exemple de code suivant montre comment utiliser `SendEmail`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email from.</
param>
    /// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
    /// <param name="subject">The subject of the email.</param>
    /// <param name="htmlContent">The HTML content of the email.</param>
    /// <param name="textContent">The text content of the email.</param>
    /// <param name="templateName">The name of the email template to use
(optional).</param>
    /// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
    /// <param name="contactListName">The name of the contact list for unsubscribe
functionality (optional).</param>
    /// <returns>The MessageId response from the SendEmail operation.</returns>
    public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,
        string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
    {
        var request = new SendEmailRequest
        {
            FromEmailAddress = fromEmailAddress
        };

        if (toEmailAddresses.Any())
        {
            request.Destination = new Destination { ToAddresses =
toEmailAddresses };
        }

        if (!string.IsNullOrEmpty(templateName))

```

```

    {
        request.Content = new EmailContent()
        {
            Template = new Template
            {
                TemplateName = templateName,
                TemplateData = templateData
            }
        };
    }
    else
    {
        request.Content = new EmailContent
        {
            Simple = new Message
            {
                Subject = new Content { Data = subject },
                Body = new Body
                {
                    Html = new Content { Data = htmlContent },
                    Text = new Content { Data = textContent }
                }
            }
        };
    }

    if (!string.IsNullOrEmpty(contactListName))
    {
        request.ListManagementOptions = new ListManagementOptions
        {
            ContactListName = contactListName
        };
    }

    try
    {
        var response = await _sesClient.SendEmailAsync(request);
        return response.MessageId;
    }
    catch (AccountSuspendedException ex)
    {
        Console.WriteLine("The account's ability to send email has been
permanently restricted.");
        Console.WriteLine(ex.Message);
    }
}

```

```
    }
    catch (MailFromDomainNotVerifiedException ex)
    {
        Console.WriteLine("The sending domain is not verified.");
        Console.WriteLine(ex.Message);
    }
    catch (MessageRejectedException ex)
    {
        Console.WriteLine("The message content is invalid.");
        Console.WriteLine(ex.Message);
    }
    catch (SendingPausedException ex)
    {
        Console.WriteLine("The account's ability to send email is currently
paused.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
    }

    return string.Empty;
}
```

- Pour plus de détails sur l'API, reportez-vous [SendEmail](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Scénario de newsletter

L'exemple de code suivant montre comment exécuter le scénario de newsletter de l'API Amazon SES v2.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez le scénario.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace Sesev2Scenario;

public static class NewsletterWorkflow
{
    /*
        This scenario demonstrates how to use the Amazon Simple Email Service (SES) v2
        to send a coupon newsletter to a list of subscribers.
        The scenario performs the following tasks:

        1. Prepare the application:
            - Create a verified email identity for sending and replying to emails.
            - Create a contact list to store the subscribers' email addresses.
            - Create an email template for the coupon newsletter.

        2. Gather subscriber email addresses:
            - Prompt the user for a base email address.
            - Create 3 variants of the email address using subaddress extensions (e.g.,
            user+sesev-weekly-newsletter-1@example.com).
            - Add each variant as a contact to the contact list.
            - Send a welcome email to each new contact.

        3. Send the coupon newsletter:
            - Retrieve the list of contacts from the contact list.
```

- Send the coupon newsletter using the email template to each contact.
4. Monitor and review:
 - Provide instructions for the user to review the sending activity and metrics in the AWS console.
 5. Clean up resources:
 - Delete the contact list (which also deletes all contacts within it).
 - Delete the email template.
 - Optionally delete the verified email identity.

*/

```

public static SESv2Wrapper _sesv2Wrapper;
public static string? _baseEmailAddress = null;
public static string? _verifiedEmail = null;
private static string _contactListName = "weekly-coupons-newsletter";
private static string _templateName = "weekly-coupons";
private static string _subject = "Weekly Coupons Newsletter";
private static string _htmlContentFile = "coupon-newsletter.html";
private static string _textContentFile = "coupon-newsletter.txt";
private static string _htmlWelcomeFile = "welcome.html";
private static string _textWelcomeFile = "welcome.txt";
private static string _couponsDataFile = "sample_coupons.json";

// Relative location of the resources folder.
private static string _resourcesFilePathLocation = "../..../resources/";

public static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonSimpleEmailServiceV2>()
                .AddTransient<SESV2Wrapper>()
        )
        .Build();

    ServicesSetup(host);

```



```

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Amazon SES v2 Coupon Newsletter
Scenario.");
            Console.WriteLine("This scenario demonstrates how to use the Amazon
Simple Email Service (SES) v2 " +
                            "\r\nto send a coupon newsletter to a list of
subscribers.");

            // Prepare the application.
            var emailIdentity = await PrepareApplication();

            // Gather subscriber email addresses.
            await GatherSubscriberEmailAddresses(emailIdentity);

            // Send the coupon newsletter.
            await SendCouponNewsletter(emailIdentity);

            // Monitor and review.
            MonitorAndReview(true);

            // Clean up resources.
            await Cleanup(emailIdentity, true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon SES v2 Coupon Newsletter scenario is
complete.");
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)

```

```

{
    _sesv2Wrapper = host.Services.GetRequiredService<SESV2Wrapper>();
}

/// <summary>
/// Set up the resources for the scenario.
/// </summary>
/// <returns>The email address of the verified identity.</returns>
public static async Task<string?> PrepareApplication()
{
    var htmlContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_htmlContentFile);
    var textContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_textContentFile);

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("1. In this step, we will prepare the application:" +
        "\r\n - Create a verified email identity for sending and
replying to emails." +
        "\r\n - Create a contact list to store the subscribers'
email addresses." +
        "\r\n - Create an email template for the coupon
newsletter.\r\n");

    // Prompt the user for a verified email address.
    while (!IsEmail(_verifiedEmail))
    {
        Console.Write("Enter a verified email address or an email to verify: ");
        _verifiedEmail = Console.ReadLine();
    }

    try
    {
        // Create an email identity and start the verification process.
        await _sesv2Wrapper.CreateEmailIdentityAsync(_verifiedEmail);
        Console.WriteLine($"Identity {_verifiedEmail} created.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Identity {_verifiedEmail} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating email identity: {ex.Message}");
    }
}

```

```

    }

    // Create a contact list.
    try
    {
        await _sesv2Wrapper.CreateContactListAsync(_contactListName);
        Console.WriteLine($"Contact list {_contactListName} created.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Contact list {_contactListName} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating contact list: {ex.Message}");
    }

    // Create an email template.
    try
    {
        await _sesv2Wrapper.CreateEmailTemplateAsync(_templateName, _subject,
htmlContent, textContent);
        Console.WriteLine($"Email template {_templateName} created.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"Email template {_templateName} already exists.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error creating email template: {ex.Message}");
    }

    return _verifiedEmail;
}

/// <summary>
/// Generate subscriber addresses and send welcome emails.
/// </summary>
/// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> GatherSubscriberEmailAddresses(string
fromEmailAddress)

```

```

{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("2. In Step 2, we will gather subscriber email addresses:"
+
        "\r\n - Prompt the user for a base email address." +
        "\r\n - Create 3 variants of the email address using
subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com)." +
        "\r\n - Add each variant as a contact to the contact
list." +
        "\r\n - Send a welcome email to each new contact.\r\n");

    // Prompt the user for a base email address.
    while (!IsEmail(_baseEmailAddress))
    {
        Console.Write("Enter a base email address (e.g., user@example.com): ");
        _baseEmailAddress = Console.ReadLine();
    }

    // Create 3 variants of the email address using +ses-weekly-newsletter-1,
+ses-weekly-newsletter-2, etc.
    var baseEmailAddressParts = _baseEmailAddress!.Split("@");
    for (int i = 1; i <= 3; i++)
    {
        string emailAddress = $"{baseEmailAddressParts[0]}+ses-weekly-
newsletter-{i}@{baseEmailAddressParts[1]}";

        try
        {
            // Create a contact with the email address in the contact list.
            await _sesv2Wrapper.CreateContactAsync(emailAddress,
            _contactListName);
            Console.WriteLine($"Contact {emailAddress} added to the
            {_contactListName} contact list.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Contact {emailAddress} already exists in the
            {_contactListName} contact list.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating contact {emailAddress}:
            {ex.Message}");
            return false;
        }
    }
}

```

```

    }

    // Send a welcome email to the new contact.
    try
    {
        string subject = "Welcome to the Weekly Coupons Newsletter";
        string htmlContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _htmlWelcomeFile);
        string textContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _textWelcomeFile);

        await _sesv2Wrapper.SendEmailAsync(fromEmailAddress, new
List<string> { emailAddress }, subject, htmlContent, textContent);
        Console.WriteLine($"Welcome email sent to {emailAddress}.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending welcome email to {emailAddress}:
{ex.Message}");
        return false;
    }

    // Wait 2 seconds before sending the next email (if the account is in
the SES Sandbox).
    await Task.Delay(2000);
}

return true;
}

/// <summary>
/// Send the coupon newsletter to the subscribers in the contact list.
/// </summary>
/// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> SendCouponNewsletter(string fromEmailAddress)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("3. In this step, we will send the coupon newsletter:" +
        "\r\n - Retrieve the list of contacts from the contact
list." +
        "\r\n - Send the coupon newsletter using the email
template to each contact.\r\n");

```

```

        // Retrieve the list of contacts from the contact list.
        var contacts = await _sesv2Wrapper.ListContactsAsync(_contactListName);
        if (!contacts.Any())
        {
            Console.WriteLine($"No contacts found in the {_contactListName} contact
list.");
            return false;
        }

        // Load the coupon data from the sample_coupons.json file.
        string couponsData = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _couponsDataFile);

        // Send the coupon newsletter to each contact using the email template.
        try
        {
            foreach (var contact in contacts)
            {
                // To use the Contact List for list management, send to only one
address at a time.
                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress,
                    new List<string> { contact.EmailAddress },
                    null, null, null, _templateName, couponsData, _contactListName);
            }

            Console.WriteLine($"Coupon newsletter sent to contact list
{_contactListName}.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error sending coupon newsletter to contact list
{_contactListName}: {ex.Message}");
            return false;
        }

        return true;
    }

    /// <summary>
    /// Provide instructions for monitoring sending activity and metrics.
    /// </summary>
    /// <param name="interactive">True to run in interactive mode.</param>

```

```

/// <returns>True if successful.</returns>
public static bool MonitorAndReview(bool interactive)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("4. In step 4, we will monitor and review:" +
        "\r\n - Provide instructions for the user to review the
sending activity and metrics in the AWS console.\r\n");

    Console.WriteLine("Review your sending activity using the SES Homepage in
the AWS console.");
    Console.WriteLine("Press Enter to open the SES Homepage in your default
browser...");
    if (interactive)
    {
        Console.ReadLine();
        try
        {
            // Open the SES Homepage in the default browser.
            Process.Start(new ProcessStartInfo
            {
                FileName = "https://console.aws.amazon.com/ses/home",
                UseShellExecute = true
            });
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error opening the SES Homepage: {ex.Message}");
            return false;
        }
    }

    Console.WriteLine("Review the sending activity and email metrics, then press
Enter to continue...");
    if (interactive)
        Console.ReadLine();
    return true;
}

/// <summary>
/// Clean up the resources used in the scenario.
/// </summary>
/// <param name="verifiedEmailAddress">The verified email address from
PrepareApplication.</param>
/// <param name="interactive">True if interactive.</param>

```

```

    /// <returns>Async task.</returns>
    public static async Task<bool> Cleanup(string verifiedEmailAddress, bool
interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Finally, we clean up resources:" +
            "\r\n - Delete the contact list (which also deletes all
contacts within it)." +
            "\r\n - Delete the email template." +
            "\r\n - Optionally delete the verified email identity.\r
\n");

        Console.WriteLine("Cleaning up resources...");

        // Delete the contact list (this also deletes all contacts in the list).
        try
        {
            await _sesv2Wrapper.DeleteContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Contact list {_contactListName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting contact list {_contactListName}:
{ex.Message}");
            return false;
        }

        // Delete the email template.
        try
        {
            await _sesv2Wrapper.DeleteEmailTemplateAsync(_templateName);
            Console.WriteLine($"Email template {_templateName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Email template {_templateName} not found.");
        }
        catch (Exception ex)
        {

```



```

        Console.WriteLine($"Error deleting email template {_templateName}:
{ex.Message}");
        return false;
    }

    // Ask the user if they want to delete the email identity.
    var deleteIdentity = !interactive ||
        GetYesNoResponse(
            $"Do you want to delete the email identity {verifiedEmailAddress}?
(y/n) ");
    if (deleteIdentity)
    {
        try
        {
            await _sesv2Wrapper.DeleteEmailIdentityAsync(verifiedEmailAddress);
            Console.WriteLine($"Email identity {verifiedEmailAddress}
deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine(
                $"Email identity {verifiedEmailAddress} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine(
                $"Error deleting email identity {verifiedEmailAddress}:
{ex.Message}");
            return false;
        }
    }
    else
    {
        Console.WriteLine(
            $"Skipping deletion of email identity {verifiedEmailAddress}.");
    }

    return true;
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>

```

```

    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Simple check to verify a string is an email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email.</returns>
    private static bool IsEmail(string? email)
    {
        if (string.IsNullOrEmpty(email))
            return false;
        return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$",
RegexOptions.IgnoreCase);
    }
}

```

Encapsuleur pour les opérations de service.

```

using System.Net;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;

namespace Sesev2Scenario;

/// <summary>
/// Wrapper class for Amazon Simple Email Service (SES) v2 operations.
/// </summary>
public class SESv2Wrapper
{
    private readonly IAmazonSimpleEmailServiceV2 _sesClient;

    /// <summary>
    /// Constructor for the SESv2Wrapper.

```

```
/// </summary>
/// <param name="sesClient">The injected SES v2 client.</param>
public SESv2Wrapper(IAmazonSimpleEmailServiceV2 sesClient)
{
    _sesClient = sesClient;
}

/// <summary>
/// Creates a contact and adds it to the specified contact list.
/// </summary>
/// <param name="emailAddress">The email address of the contact.</param>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The response from the CreateContact operation.</returns>
public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
{
    var request = new CreateContactRequest
    {
        EmailAddress = emailAddress,
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
        Console.WriteLine(ex.Message);
        return true;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
    }
}
```

```

        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
    }
    return false;
}

/// <summary>
/// Creates a contact list with the specified name.
/// </summary>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateContactListAsync(string contactListName)
{
    var request = new CreateContactListRequest
    {
        ContactListName = contactListName
    };

    try
    {
        var response = await _sesClient.CreateContactListAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Contact list with name {contactListName} already
exists.");
        Console.WriteLine(ex.Message);
        return true;
    }
    catch (LimitExceededException ex)
    {
        Console.WriteLine("The limit for contact lists has been exceeded.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
}

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
    }
    return false;
}

/// <summary>
/// Creates an email identity (email address or domain) and starts the
verification process.
/// </summary>
/// <param name="emailIdentity">The email address or domain to create and
verify.</param>
/// <returns>The response from the CreateEmailIdentity operation.</returns>
public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
{
    var request = new CreateEmailIdentityRequest
    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.CreateEmailIdentityAsync(request);
        return response;
    }
    catch (AlreadyExistsException ex)
    {
        Console.WriteLine($"Email identity {emailIdentity} already exists.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (LimitExceededException ex)
    {

```

```

        Console.WriteLine("The limit for email identities has been exceeded.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Creates an email template with the specified content.
/// </summary>
/// <param name="templateName">The name of the email template.</param>
/// <param name="subject">The subject of the email template.</param>
/// <param name="htmlContent">The HTML content of the email template.</param>
/// <param name="textContent">The text content of the email template.</param>
/// <returns>True if successful.</returns>
public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
{
    var request = new CreateEmailTemplateRequest
    {
        TemplateName = templateName,
        TemplateContent = new EmailTemplateContent
        {
            Subject = subject,
            Html = htmlContent,

```

```

        Text = textContent
    }
};

try
{
    var response = await _sesClient.CreateEmailTemplateAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}
catch (AlreadyExistsException ex)
{
    Console.WriteLine($"Email template with name {templateName} already
exists.");
    Console.WriteLine(ex.Message);
}
catch (LimitExceededException ex)
{
    Console.WriteLine("The limit for email templates has been exceeded.");
    Console.WriteLine(ex.Message);
}
catch (TooManyRequestsException ex)
{
    Console.WriteLine("Too many requests were made. Please try again
later.");
    Console.WriteLine(ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while creating the email template:
{ex.Message}");
}

return false;
}

/// <summary>
/// Deletes a contact list and all contacts within it.
/// </summary>
/// <param name="contactListName">The name of the contact list to delete.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteContactListAsync(string contactListName)
{
    var request = new DeleteContactListRequest

```

```

        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest

```



```

    {
        EmailIdentity = emailIdentity
    };

    try
    {
        var response = await _sesClient.DeleteEmailIdentityAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (ConcurrentModificationException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
        Console.WriteLine(ex.Message);
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
    }

    return false;
}

/// <summary>
/// Deletes an email template.
/// </summary>
/// <param name="templateName">The name of the email template to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteEmailTemplateAsync(string templateName)
{
    var request = new DeleteEmailTemplateRequest

```

```

        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {

```

```

        var response = await _sesClient.ListContactsAsync(request);
        return response.Contacts;
    }
    catch (NotFoundException ex)
    {
        Console.WriteLine($"The contact list {contactListName} does not
exist.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
    }

    return new List<Contact>();
}

/// <summary>
/// Sends an email with the specified content and options.
/// </summary>
/// <param name="fromEmailAddress">The email address to send the email from.</
param>
/// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
/// <param name="subject">The subject of the email.</param>
/// <param name="htmlContent">The HTML content of the email.</param>
/// <param name="textContent">The text content of the email.</param>
/// <param name="templateName">The name of the email template to use
(optional).</param>
/// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
/// <param name="contactListName">The name of the contact list for unsubscribe
functionality (optional).</param>
/// <returns>The MessageId response from the SendEmail operation.</returns>
public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,

```

```
        string? htmlContent, string? textContent, string? templateName = null,
        string? templateData = null, string? contactListName = null)
    {
        var request = new SendEmailRequest
        {
            FromEmailAddress = fromEmailAddress
        };

        if (toEmailAddresses.Any())
        {
            request.Destination = new Destination { ToAddresses =
toEmailAddresses };
        }

        if (!string.IsNullOrEmpty(templateName))
        {
            request.Content = new EmailContent()
            {
                Template = new Template
                {
                    TemplateName = templateName,
                    TemplateData = templateData
                }
            };
        }
        else
        {
            request.Content = new EmailContent
            {
                Simple = new Message
                {
                    Subject = new Content { Data = subject },
                    Body = new Body
                    {
                        Html = new Content { Data = htmlContent },
                        Text = new Content { Data = textContent }
                    }
                }
            };
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
```

```
        {
            ContactListName = contactListName
        };
    }

    try
    {
        var response = await _sesClient.SendEmailAsync(request);
        return response.MessageId;
    }
    catch (AccountSuspendedException ex)
    {
        Console.WriteLine("The account's ability to send email has been
permanently restricted.");
        Console.WriteLine(ex.Message);
    }
    catch (MailFromDomainNotVerifiedException ex)
    {
        Console.WriteLine("The sending domain is not verified.");
        Console.WriteLine(ex.Message);
    }
    catch (MessageRejectedException ex)
    {
        Console.WriteLine("The message content is invalid.");
        Console.WriteLine(ex.Message);
    }
    catch (SendingPausedException ex)
    {
        Console.WriteLine("The account's ability to send email is currently
paused.");
        Console.WriteLine(ex.Message);
    }
    catch (TooManyRequestsException ex)
    {
        Console.WriteLine("Too many requests were made. Please try again
later.");
        Console.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
    }
}
```

```
        return string.Empty;
    }
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateContact](#)
 - [CreateContactList](#)
 - [CreateEmailIdentity](#)
 - [CreateEmailTemplate](#)
 - [DeleteContactList](#)
 - [DeleteEmailIdentity](#)
 - [DeleteEmailTemplate](#)
 - [ListContacts](#)
 - [SendEmail.simple](#)
 - [SendEmail.modèle](#)

Exemples d'utilisation d'Amazon SNS SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon SNS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)

- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)

Mise en route

Hello Amazon SNS

L'exemple de code suivant montre comment commencer à utiliser Amazon SNS.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
```

```

        Console.WriteLine($"\\tTopic ARN: {topic.TopicArn}");
        Console.WriteLine();
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [ListTopics](#) à la section Référence des AWS SDK pour .NET API.

Actions

CheckIfPhoneNumberIsOptedOut

L'exemple de code suivant montre comment utiliser `CheckIfPhoneNumberIsOptedOut`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use the Amazon Simple Notification Service
/// (Amazon SNS) to check whether a phone number has been opted out.
/// </summary>
public class IsPhoneNumOptedOut
{
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

```



```

        await CheckIfOptedOutAsync(client, phoneNumber);
    }

    /// <summary>
    /// Checks to see if the supplied phone number has been opted out.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS Client object used
    /// to check if the phone number has been opted out.</param>
    /// <param name="phoneNumber">A string representing the phone number
    /// to check.</param>
    public static async Task
    CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
    {
        var request = new CheckIfPhoneNumberIsOptedOutRequest
        {
            PhoneNumber = phoneNumber,
        };

        try
        {
            var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                string optOutStatus = response.IsOptedOut ? "opted out" : "not
opted out.";
                Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
            }
        }
        catch (AuthorizationException ex)
        {
            Console.WriteLine($"{ex.Message}");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CheckIfPhoneNumberIsOptedOut](#) à la section Référence des AWS SDK pour .NET API.

CreateTopic

L'exemple de code suivant montre comment utiliser `CreateTopic`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une rubrique avec un nom spécifique.

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use Amazon Simple Notification Service
/// (Amazon SNS) to add a new Amazon SNS topic.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}");
    }

    /// <summary>
    /// Creates a new SNS topic using the supplied topic name.
    /// </summary>
    /// <param name="client">The initialized SNS client object used to
    /// create the new topic.</param>
    /// <param name="topicName">A string representing the topic name.</param>
    /// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>
}
```

```

    public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
    {
        var request = new CreateTopicRequest
        {
            Name = topicName,
        };

        var response = await client.CreateTopicAsync(request);

        return response.TopicArn;
    }
}

```

Créez une nouvelle rubrique avec un nom et des attributs FIFO et de déduplication spécifiques.

```

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
    attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
    duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
    useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }
        }
    }
}

```

```

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
            { "FifoTopic", "true" }
        };
        if (useContentBasedDeduplication)
        {
            createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
        }
    }

    var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
    return createResponse.TopicArn;
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateTopic](#) à la section Référence des AWS SDK pour .NET API.

DeleteTopic

L'exemple de code suivant montre comment utiliser `DeleteTopic`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Supprimez une rubrique à l'aide de son ARN de rubrique.

```

/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{

```

```
var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(  
    new DeleteTopicRequest()  
    {  
        TopicArn = topicArn  
    });  
return deleteResponse.HttpStatusCode == HttpStatusCode.OK;  
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteTopic](#) à la section Référence des AWS SDK pour .NET API.

GetTopicAttributes

L'exemple de code suivant montre comment utiliser `GetTopicAttributes`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Amazon.SimpleNotificationService;  
  
/// <summary>  
/// This example shows how to retrieve the attributes of an Amazon Simple  
/// Notification Service (Amazon SNS) topic.  
/// </summary>  
public class GetTopicAttributes  
{  
    public static async Task Main()  
    {  
        string topicArn = "arn:aws:sns:us-west-2:000000000000:ExampleSNSTopic";  
        IAmazonSimpleNotificationService client = new  
AmazonSimpleNotificationServiceClient();
```

```

        var attributes = await GetTopicAttributesAsync(client, topicArn);
        DisplayTopicAttributes(attributes);
    }

    /// <summary>
    /// Given the ARN of the Amazon SNS topic, this method retrieves the topic
    /// attributes.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the attributes for the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic for which to retrieve
    /// the attributes.</param>
    /// <returns>A Dictionary of topic attributes.</returns>
    public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
    {
        var response = await client.GetTopicAttributesAsync(topicArn);

        return response.Attributes;
    }

    /// <summary>
    /// This method displays the attributes for an Amazon SNS topic.
    /// </summary>
    /// <param name="topicAttributes">A Dictionary containing the
    /// attributes for an Amazon SNS topic.</param>
    public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
    {
        foreach (KeyValuePair<string, string> entry in topicAttributes)
        {
            Console.WriteLine($"{entry.Key}: {entry.Value}\n");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [GetTopicAttributes](#) à la section Référence des AWS SDK pour .NET API.

ListSubscriptions

L'exemple de code suivant montre comment utiliser `ListSubscriptions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example will retrieve a list of the existing Amazon Simple
/// Notification Service (Amazon SNS) subscriptions.
/// </summary>
public class ListSubscriptions
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        Console.WriteLine("Enter a topic ARN to list subscriptions for a
specific topic, " +
                        "or press Enter to list subscriptions for all
topics.");
        var topicArn = Console.ReadLine();
        Console.WriteLine();

        var subscriptions = await GetSubscriptionsListAsync(client, topicArn);

        DisplaySubscriptionList(subscriptions);
    }

    /// <summary>
```

```

    /// Gets a list of the existing Amazon SNS subscriptions, optionally by
    specifying a topic ARN.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to obtain the list of subscriptions.</param>
    /// <param name="topicArn">The optional ARN of a specific topic. Defaults to
    null.</param>
    /// <returns>A list containing information about each subscription.</
returns>
    public static async Task<List<Subscription>>
    GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string topicArn
    = null)
    {
        var results = new List<Subscription>();

        if (!string.IsNullOrEmpty(topicArn))
        {
            var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
                new ListSubscriptionsByTopicRequest()
                {
                    TopicArn = topicArn,
                });

            // Get the entire list using the paginator.
            await foreach (var subscription in paginateByTopic.Subscriptions)
            {
                results.Add(subscription);
            }
        }
        else
        {
            var paginateAllSubscriptions =
            client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());

            // Get the entire list using the paginator.
            await foreach (var subscription in
            paginateAllSubscriptions.Subscriptions)
            {
                results.Add(subscription);
            }
        }

        return results;
    }

```



```
/// <summary>
/// Display a list of Amazon SNS subscription information.
/// </summary>
/// <param name="subscriptionList">A list containing details for existing
/// Amazon SNS subscriptions.</param>
public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
{
    foreach (var subscription in subscriptionList)
    {
        Console.WriteLine($"Owner: {subscription.Owner}");
        Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
        Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
        Console.WriteLine($"Endpoint: {subscription.Endpoint}");
        Console.WriteLine($"Protocol: {subscription.Protocol}");
        Console.WriteLine();
    }
}
}
```

- Pour plus de détails sur l'API, reportez-vous [ListSubscriptions](#) à la section Référence des AWS SDK pour .NET API.

ListTopics

L'exemple de code suivant montre comment utiliser `ListTopics`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
```

```
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// Lists the Amazon Simple Notification Service (Amazon SNS)
/// topics for the current account.
/// </summary>
public class ListSNSTopics
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await GetTopicListAsync(client);
    }

    /// <summary>
    /// Retrieves the list of Amazon SNS topics in groups of up to 100
    /// topics.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the list of topics.</param>
    public static async Task GetTopicListAsync(IAmazonSimpleNotificationService
client)
    {
        // If there are more than 100 Amazon SNS topics, the call to
        // ListTopicsAsync will return a value to pass to the
        // method to retrieve the next 100 (or less) topics.
        string nextToken = string.Empty;

        do
        {
            var response = await client.ListTopicsAsync(nextToken);
            DisplayTopicsList(response.Topics);
            nextToken = response.NextToken;
        }
        while (!string.IsNullOrEmpty(nextToken));
    }

    /// <summary>
    /// Displays the list of Amazon SNS Topic ARNs.
    /// </summary>
```

```
/// <param name="topicList">The list of Topic ARNs.</param>
public static void DisplayTopicsList(List<Topic> topicList)
{
    foreach (var topic in topicList)
    {
        Console.WriteLine($"{topic.TopicArn}");
    }
}
}
```

- Pour plus de détails sur l'API, reportez-vous [ListTopics](#) à la section Référence des AWS SDK pour .NET API.

Publish

L'exemple de code suivant montre comment utiliser `Publish`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Publier un message dans une rubrique

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example publishes a message to an Amazon Simple Notification
/// Service (Amazon SNS) topic.
/// </summary>
public class PublishToSNSTopic
{
    public static async Task Main()
    {

```

```

        string topicArn = "arn:aws:sns:us-east-2:000000000000:ExampleSNSTopic";
        string messageText = "This is an example message to publish to the
ExampleSNSTopic.";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await PublishToTopicAsync(client, topicArn, messageText);
    }

    /// <summary>
    /// Publishes a message to an Amazon SNS topic.
    /// </summary>
    /// <param name="client">The initialized client object used to publish
    /// to the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="messageText">The text of the message.</param>
    public static async Task PublishToTopicAsync(
        IAmazonSimpleNotificationService client,
        string topicArn,
        string messageText)
    {
        var request = new PublishRequest
        {
            TopicArn = topicArn,
            Message = messageText,
        };

        var response = await client.PublishAsync(request);

        Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
    }
}

```

Publiez un message dans une rubrique avec des options de groupe, de duplication et d'attribut.

```

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>

```

```

public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                             "\r\nAll messages within the same group will be
received in the order " +
                             "they were published.");

            Console.WriteLine();
            var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
                                 "you must enter a deduplication ID.");

                Console.WriteLine("Enter a deduplication ID for this message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
                Console.WriteLine("Enter a number for an attribute.");
                for (int i = 0; i < _tones.Length; i++)
                {
                    Console.WriteLine($"{i + 1}. {_tones[i]}");
                }
            }
        }
    }
}

```

```

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }

        var messageID = await SnsWrapper.PublishToTopicWithAttribute(
            _topicArn, message, "tone", toneAttribute, deduplicationId,
            messageGroupId);

        Console.WriteLine($"Message published with id {messageID}.");
    }

    keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

```

Appliquez les sélections de l'utilisateur à l'action de publication.

```

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
    and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)

```

```
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
            };
    }

    var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}
```

- Pour plus de détails sur l'API, consultez [Publish](#) dans la Référence des API du kit AWS SDK pour .NET .

Subscribe

L'exemple de code suivant montre comment utiliser `Subscribe`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Abonnez une adresse e-mail à une rubrique.

```

    /// <summary>
    /// Creates a new subscription to a topic.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object, used
    /// to create an Amazon SNS subscription.</param>
    /// <param name="topicArn">The ARN of the topic to subscribe to.</param>
    /// <returns>A SubscribeResponse object which includes the subscription
    /// ARN for the new subscription.</returns>
    public static async Task<SubscribeResponse> TopicSubscribeAsync(
        IAmazonSimpleNotificationService client,
        string topicArn)
    {
        SubscribeRequest request = new SubscribeRequest()
        {
            TopicArn = topicArn,
            ReturnSubscriptionArn = true,
            Protocol = "email",
            Endpoint = "recipient@example.com",
        };

        var response = await client.SubscribeAsync(request);

        return response;
    }

```

Abonnez une file d'attente à une rubrique avec des filtres facultatifs.

```

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
    subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
    filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {

```



```

        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
        { { "FilterPolicy", filterPolicy } };
    }

    var subscribeResponse = await
        _amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}

```

- Pour plus de détails sur l'API, consultez [Subscribe](#) dans la Référence des API du kit AWS SDK pour .NET .

Unsubscribe

L'exemple de code suivant montre comment utiliser `Unsubscribe`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Désabonnez-vous d'une rubrique à l'aide d'un ARN d'abonnement.

```

/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{

```

```
var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(  
    new UnsubscribeRequest()  
    {  
        SubscriptionArn = subscriptionArn  
    });  
return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;  
}
```

- Pour plus de détails sur l'API, consultez [Unsubscribe](#) dans la Référence des API du kit AWS SDK pour .NET .

Scénarios

Création d'une application Amazon SNS

L'exemple de code suivant montre comment créer une application dotée de fonctionnalités d'abonnement et de publication et qui traduit les messages.

SDK pour .NET

Indique comment utiliser l'API .NET Amazon Simple Notification Service pour créer une application Web dotée de fonctionnalités d'abonnement et de publication. De plus, cet exemple d'application traduit également des messages.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Amazon SNS
- Amazon Translate

Création d'une application sans serveur pour gérer des photos

L'exemple de code suivant montre comment créer une application sans serveur permettant aux utilisateurs de gérer des photos à l'aide d'étiquettes.

SDK pour .NET

Montre comment développer une application de gestion de ressources photographiques qui détecte les étiquettes dans les images à l'aide d'Amazon Rekognition et les stocke pour les récupérer ultérieurement.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Pour explorer en profondeur l'origine de cet exemple, consultez l'article sur [AWS Community](#).

Les services utilisés dans cet exemple

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Publier un message texte SMS

L'exemple de code suivant montre comment publier des messages texte à l'aide d'Amazon SNS.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;
```

```
public class SNSMessage
{
    private AmazonSimpleNotificationServiceClient snsClient;

    /// <summary>
    /// Initializes a new instance of the <see cref="SNSMessage"/> class.
    /// Constructs a new SNSMessage object initializing the Amazon Simple
    /// Notification Service (Amazon SNS) client using the supplied
    /// Region endpoint.
    /// </summary>
    /// <param name="regionEndpoint">The Amazon Region endpoint to use in
    /// sending test messages with this object.</param>
    public SNSMessage(RegionEndpoint regionEndpoint)
    {
        snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);
    }

    /// <summary>
    /// Sends the SMS message passed in the text parameter to the phone number
    /// in phoneNum.
    /// </summary>
    /// <param name="phoneNum">The ten-digit phone number to which the text
    /// message will be sent.</param>
    /// <param name="text">The text of the message to send.</param>
    /// <returns>Async task.</returns>
    public async Task SendTextMessageAsync(string phoneNum, string text)
    {
        if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
        {
            return;
        }

        // Now actually send the message.
        var request = new PublishRequest
        {
            Message = text,
            PhoneNumber = phoneNum,
        };

        try
        {
            var response = await snsClient.PublishAsync(request);
        }
        catch (Exception ex)
```

```

        {
            Console.WriteLine($"Error sending message: {ex}");
        }
    }
}

```

- Pour plus d'informations sur l'API, consultez [Publier](#) dans AWS SDK pour .NET Référence de l'API.

Publier des messages dans des files d'attente

L'exemple de code suivant illustre comment :

- Créer une rubrique (FIFO ou non FIFO).
- Abonner plusieurs files d'attente à la rubrique avec la possibilité d'appliquer un filtre.
- Publier des messages dans la rubrique.
- Interroger les files d'attente pour les messages reçus.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```

/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;
}

```

```

private static readonly int _queueCount = 2;
private static readonly string[] _queueUrls = new string[_queueCount];
private static readonly string[] _subscriptionArns = new string[_queueCount];
private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
public static SNSWrapper SnsWrapper { get; set; } = null!;
public static SQSWrapper SqsWrapper { get; set; } = null!;
public static bool UseConsole { get; set; } = true;
static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EventBridge.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonSQS>()
                .AddAWSService<IAmazonSimpleNotificationService>()
                .AddTransient<SNSWrapper>()
                .AddTransient<SQSWrapper>()
        )
        .Build();

    ServicesSetup(host);
    PrintDescription();

    await RunScenario();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
    SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
}

/// <summary>

```

```

    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
            $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
            $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
            $"\r\nFIFO topics deliver messages in order and support
deduplication and message filtering." +
            $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
"example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
is supported." +

```



```

        $"\\r\\nDeduplication IDs are either set in the message
or automatically generated " +
        $"\\r\\nfrom content using a hash function.\\r\\n" +
        $"\\r\\nIf a message is successfully published to an SNS
FIFO topic, any message " +
        $"\\r\\npublished and determined to have the same
deduplication ID, " +
        $"\\r\\nwithin the five-minute deduplication interval,
is accepted but not delivered.\\r\\n" +
        $"\\r\\nFor more information about deduplication, " +
        $"\\r\\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

        _useContentBasedDeduplication = GetYesNoResponse("Use content-based
deduplication instead of entering a deduplication ID?");
        Console.WriteLine(new string('-', 80));
    }

    _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
_useContentBasedDeduplication);

    Console.WriteLine($"Your new topic with the name {_topicName}" +
        $"\\r\\nand Amazon Resource Name (ARN) {_topicArn}" +
        $"\\r\\nhas been created.\\r\\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)
    {
        var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
    }
}

```

```

        if (_useFifoTopic)
        {
            // Only explain this once.
            if (i == 0)
            {
                Console.WriteLine(
                    "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
            }

            var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

            _queueUrls[i] = queueUrl;

            Console.WriteLine($"Your new queue with the name {queueName}" +
                $"\r\nand queue URL {queueUrl}" +
                $"\r\nhas been created.\r\n");

            if (i == 0)
            {
                Console.WriteLine(
                    $"The queue URL is used to retrieve the queue ARN,\r\n" +
                    $"which is used to create a subscription.");
                Console.WriteLine(new string('-', 80));
            }

            var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

            if (i == 0)
            {
                Console.WriteLine(
                    $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                    $"messages from an SNS topic");
            }

            await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

            await SetupFilters(i, queueArn, queueName);
        }
    }
}

```

```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a " +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
        }
    }

```

```

        var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
        queueArn);
        _subscriptionArns[queueCount] = subscriptionArn;

        Console.WriteLine(
            $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
            $"with the subscription ARN {subscriptionArn}");
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
        int.TryParse(selection, out selectionNumber);
        if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
        {
            filterSelections.Add(_tones[selectionNumber - 1]);
        }
    }
}

```

```

    } while (selectionNumber != 0);

    var filters = new Dictionary<string, List<string>>
    {
        { "tone", filterSelections }
    };
    string filterPolicy = JsonSerializer.Serialize(filters);
    return filterPolicy;
}

/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set a message group ID." +
                               "\r\nAll messages within the same group will be received in the order " +
                               "they were published.");

            Console.WriteLine();
            var messageId = GetUserResponse("Enter a message group ID for this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based deduplication, " +
                                   "you must enter a deduplication ID.");
            }
        }
    }
}

```

```

        Console.WriteLine("Enter a deduplication ID for this message.");
        deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
    }

    if (GetYesNoResponse("Add an attribute to this message?"))
    {
        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageId = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
}

```

```

        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                               $"\\n\\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

```

```
/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CleanupResources()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    try
    {
        foreach (var queueUrl in _queueUrls)
        {
            if (!string.IsNullOrEmpty(queueUrl))
            {
                var deleteQueue =
                    GetYesNoResponse($"Delete queue with url {queueUrl}?");
                if (deleteQueue)
                {
                    await SqsWrapper.DeleteQueueByUrl(queueUrl);
                }
            }
        }

        foreach (var subscriptionArn in _subscriptionArns)
        {
            if (!string.IsNullOrEmpty(subscriptionArn))
            {
                await SnsWrapper.UnsubscribeByArn(subscriptionArn);
            }
        }

        var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
        if (deleteTopic)
        {
            await SnsWrapper.DeleteTopicByArn(_topicArn);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
    }
}
```



```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                ynResponse.Equals("y",
                    StringComparison.InvariantCultureIgnoreCase);

            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

```

```
}
}
```

Créez une classe qui englobe les opérations Amazon SQS.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
```

```

        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
        createQueueRequest.Attributes.Add(
            QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

    return getAttributesResponse.QueueARN;
}

```

```

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
            "\"Version\": \"2012-10-17\", " +
            "\"Statement\": [{" +
                "\"Effect\": \"Allow\", " +
                "\"Principal\": { " +
                    "\"Service\": " +
                        "\"sns.amazonaws.com\"" +
                    "}, " +
                "\"Action\": \"sqs:SendMessage\", " +
                "\"Resource\": \"{queueArn}\", " +
                "\"Condition\": { " +
                    "\"ArnEquals\": { " +
                        "\"aws:SourceArn\": \"{topicArn}\""
+
                    "}" +
                "}" +
            "}] " +
        "};

        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>

```

```

    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

```

```

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}

```

Créez une classe qui englobe les opérations Amazon SNS.

```

    /// <summary>
    /// Wrapper for Amazon Simple Notification Service (SNS) operations.
    /// </summary>
    public class SNSWrapper
    {
        private readonly IAmazonSimpleNotificationService _amazonSNSClient;

        /// <summary>
        /// Constructor for the Amazon SNS wrapper.
        /// </summary>
        /// <param name="amazonSNS">The injected Amazon SNS client.</param>
        public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
        {
            _amazonSNSClient = amazonSNS;
        }

        /// <summary>
        /// Create a new topic with a name and specific FIFO and de-duplication
        attributes.
        /// </summary>
        /// <param name="topicName">The name for the topic.</param>
        /// <param name="useFifoTopic">True to use a FIFO topic.</param>

```

```

    /// <param name="useContentBasedDeduplication">True to use content-based de-
    duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
    useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
        _amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
    subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>

```

```

    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {

```



```

        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
            };
    }

    var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}

/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{

```

```
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(  
            new DeleteTopicRequest()  
            {  
                TopicArn = topicArn  
            });  
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;  
    }  
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Exemples sans serveur

Invocation d'une fonction lambda à partir d'un déclencheur Amazon SNS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une rubrique SNS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Consommation d'un événement SNS avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
    }
}
```

```
    }  
    catch (Exception e)  
    {  
        //You can use Dead Letter Queue to handle failures. By configuring a  
        Lambda DLQ.  
        context.Logger.LogError($"An error occurred");  
        throw;  
    }  
}  
}
```

Exemples d'utilisation d'Amazon SQS SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon SQS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Actions](#)
- [Scénarios](#)
- [Exemples sans serveur](#)

Mise en route

Bonjour Amazon SQS

L'exemple de code suivant montre comment commencer à utiliser Amazon SQS.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [ListQueues](#) à la section Référence des AWS SDK pour .NET API.

Actions

CreateQueue

L'exemple de code suivant montre comment utiliser `CreateQueue`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une file d'attente avec un nom spécifique.

```
/// <summary>
/// Create a queue with a specific name.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <param name="useFifoQueue">True to use a FIFO queue.</param>
/// <returns>The url for the queue.</returns>
public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
{
    int maxMessage = 256 * 1024;
    var queueAttributes = new Dictionary<string, string>
    {
        {
            QueueAttributeName.MaximumMessageSize,
            maxMessage.ToString()
        }
    };

    var createQueueRequest = new CreateQueueRequest()
    {
        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
```

```

        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
        createQueueRequest.Attributes.Add(
            QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

```

Créez une file d'attente Amazon SQS et envoyez-lui un message.

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
    // Specify your AWS Region (an example Region is shown).
    private static readonly string QueueName = "Example_Queue";
    private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
    private static IAmazonSQS client;

    public static async Task Main()
    {
        client = new AmazonSQSClient(ServiceRegion);
        var createQueueResponse = await CreateQueue(client, QueueName);

        string queueUrl = createQueueResponse.QueueUrl;
    }
}

```

```

        Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
    {
        { "Title",    new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
        { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
        { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
    };

    string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";

    var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
messageAttributes);
}

/// <summary>
/// Creates a new Amazon SQS queue using the queue name passed to it
/// in queueName.
/// </summary>
/// <param name="client">An SQS client object used to send the message.</
param>
/// <param name="queueName">A string representing the name of the queue
/// to create.</param>
/// <returns>A CreateQueueResponse that contains information about the
/// newly created queue.</returns>
public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
string queueName)
{
    var request = new CreateQueueRequest
    {
        QueueName = queueName,
        Attributes = new Dictionary<string, string>
        {
            { "DelaySeconds", "60" },
            { "MessageRetentionPeriod", "86400" },
        },
    };

    var response = await client.CreateQueueAsync(request);
    Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");
}

```



```

        return response;
    }

    /// <summary>
    /// Sends a message to an SQS queue.
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueUrl">The URL of the queue to which to send the
    /// message.</param>
    /// <param name="messageBody">A string representing the body of the
    /// message to be sent to the queue.</param>
    /// <param name="messageAttributes">Attributes for the message to be
    /// sent to the queue.</param>
    /// <returns>A SendMessageResponse object that contains information
    /// about the message that was sent.</returns>
    public static async Task<SendMessageResponse> SendMessage(
        IAmazonSQS client,
        string queueUrl,
        string messageBody,
        Dictionary<string, MessageAttributeValue> messageAttributes)
    {
        var sendMessageRequest = new SendMessageRequest
        {
            DelaySeconds = 10,
            MessageAttributes = messageAttributes,
            MessageBody = messageBody,
            QueueUrl = queueUrl,
        };

        var response = await client.SendMessageAsync(sendMessageRequest);
        Console.WriteLine($"Sent a message with id : {response.MessageId}");

        return response;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [CreateQueue](#) à la section Référence des AWS SDK pour .NET API.

DeleteMessage

L'exemple de code suivant montre comment utiliser `DeleteMessage`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Recevez un message d'une file d'attente Amazon SQS, puis supprimez-le.

```
public static async Task Main()
{
    // If the AWS Region you want to use is different from
    // the AWS Region defined for the default user, supply
    // the specify your AWS Region to the client constructor.
    var client = new AmazonSQSClient();
    string queueName = "Example_Queue";

    var queueUrl = await GetQueueUrl(client, queueName);
    Console.WriteLine($"The SQS queue's URL is {queueUrl}");

    var response = await ReceiveAndDeleteMessage(client, queueUrl);

    Console.WriteLine($"Message: {response.Messages[0]}");
}

/// <summary>
/// Retrieve the queue URL for the queue named in the queueName
/// property using the client object.
/// </summary>
/// <param name="client">The Amazon SQS client used to retrieve the
/// queue URL.</param>
/// <param name="queueName">A string representing name of the queue
/// for which to retrieve the URL.</param>
/// <returns>The URL of the queue.</returns>
public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
{
    var request = new GetQueueUrlRequest
```

```

        {
            QueueName = queueName,
        };

        GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
        return response.QueueUrl;
    }

    /// <summary>
    /// Retrieves the message from the quque at the URL passed in the
    /// queueUrl parameters using the client.
    /// </summary>
    /// <param name="client">The SQS client used to retrieve a message.</param>
    /// <param name="queueUrl">The URL of the queue from which to retrieve
    /// a message.</param>
    /// <returns>The response from the call to ReceiveMessageAsync.</returns>
    public static async Task<ReceiveMessageResponse>
    ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
    {
        // Receive a single message from the queue.
        var receiveMessageRequest = new ReceiveMessageRequest
        {
            AttributeNames = { "SentTimestamp" },
            MaxNumberOfMessages = 1,
            MessageAttributeNames = { "All" },
            QueueUrl = queueUrl,
            VisibilityTimeout = 0,
            WaitTimeSeconds = 0,
        };

        var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

        // Delete the received message from the queue.
        var deleteMessageRequest = new DeleteMessageRequest
        {
            QueueUrl = queueUrl,
            ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
        };

        await client.DeleteMessageAsync(deleteMessageRequest);

        return receiveMessageResponse;
    }

```

```
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteMessage](#) à la section Référence des AWS SDK pour .NET API.

DeleteMessageBatch

L'exemple de code suivant montre comment utiliser `DeleteMessageBatch`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a batch of messages from a queue by its url.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
{
    var deleteRequest = new DeleteMessageBatchRequest()
    {
        QueueUrl = queueUrl,
        Entries = new List<DeleteMessageBatchRequestEntry>()
    };
    foreach (var message in messages)
    {
        deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
        {
            ReceiptHandle = message.ReceiptHandle,
            Id = message.MessageId
        });
    }
}
```

```
var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

return deleteResponse.Failed.Any();
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteMessageBatch](#) à la section Référence des AWS SDK pour .NET API.

DeleteQueue

L'exemple de code suivant montre comment utiliser `DeleteQueue`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Supprimez une file d'attente à l'aide de son URL.

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteQueue](#) à la section Référence des AWS SDK pour .NET API.

GetQueueAttributes

L'exemple de code suivant montre comment utiliser `GetQueueAttributes`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

    return getAttributesResponse.QueueARN;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetQueueAttributes](#) à la section Référence des AWS SDK pour .NET API.

GetQueueUrl

L'exemple de code suivant montre comment utiliser `GetQueueUrl`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

public class GetQueueUrl
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
    /// queue.
    /// </summary>
    public static async Task Main()
    {
        // If the Amazon SQS message queue is not in the same AWS Region as your
        // default user, you need to provide the AWS Region as a parameter to
        the
        // client constructor.
        var client = new AmazonSQSClient();

        string queueName = "New-Example-Queue";

        try
        {
            var response = await client.GetQueueUrlAsync(queueName);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"The URL for {queueName} is:
{response.QueueUrl}");
            }
        }
    }
}
```

```

    }
    catch (QueueDoesNotExistException ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine($"The queue {queueName} was not found.");
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [GetQueueUrl](#) à la section Référence des AWS SDK pour .NET API.

ReceiveMessage

L'exemple de code suivant montre comment utiliser `ReceiveMessage`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Recevez les messages d'une file d'attente à l'aide de son URL.

```

/// <summary>
/// Receive messages from a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>The list of messages.</returns>
public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
{
    // Setting WaitTimeSeconds to non-zero enables long polling.
    // For information about long polling, see
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
    SQSDeveloperGuide/sqs-short-and-long-polling.html
    var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
        new ReceiveMessageRequest()

```



```

        {
            QueueUrl = queueUrl,
            MaxNumberOfMessages = maxMessages,
            WaitTimeSeconds = 1
        });
    return messageResponse.Messages;
}

```

Recevez un message d'une file d'attente Amazon SQS, puis supprimez-le.

```

public static async Task Main()
{
    // If the AWS Region you want to use is different from
    // the AWS Region defined for the default user, supply
    // the specify your AWS Region to the client constructor.
    var client = new AmazonSQSClient();
    string queueName = "Example_Queue";

    var queueUrl = await GetQueueUrl(client, queueName);
    Console.WriteLine($"The SQS queue's URL is {queueUrl}");

    var response = await ReceiveAndDeleteMessage(client, queueUrl);

    Console.WriteLine($"Message: {response.Messages[0]}");
}

/// <summary>
/// Retrieve the queue URL for the queue named in the queueName
/// property using the client object.
/// </summary>
/// <param name="client">The Amazon SQS client used to retrieve the
/// queue URL.</param>
/// <param name="queueName">A string representing name of the queue
/// for which to retrieve the URL.</param>
/// <returns>The URL of the queue.</returns>
public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
{
    var request = new GetQueueUrlRequest
    {
        QueueName = queueName,
    };
}

```

```

        GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
        return response.QueueUrl;
    }

    /// <summary>
    /// Retrieves the message from the queue at the URL passed in the
    /// queueUrl parameters using the client.
    /// </summary>
    /// <param name="client">The SQS client used to retrieve a message.</param>
    /// <param name="queueUrl">The URL of the queue from which to retrieve
    /// a message.</param>
    /// <returns>The response from the call to ReceiveMessageAsync.</returns>
    public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
    {
        // Receive a single message from the queue.
        var receiveMessageRequest = new ReceiveMessageRequest
        {
            AttributeNames = { "SentTimestamp" },
            MaxNumberOfMessages = 1,
            MessageAttributeNames = { "All" },
            QueueUrl = queueUrl,
            VisibilityTimeout = 0,
            WaitTimeSeconds = 0,
        };

        var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

        // Delete the received message from the queue.
        var deleteMessageRequest = new DeleteMessageRequest
        {
            QueueUrl = queueUrl,
            ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
        };

        await client.DeleteMessageAsync(deleteMessageRequest);

        return receiveMessageResponse;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ReceiveMessage](#) à la section Référence des AWS SDK pour .NET API.

SendMessage

L'exemple de code suivant montre comment utiliser `SendMessage`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Créez une file d'attente Amazon SQS et envoyez-lui un message.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
    // Specify your AWS Region (an example Region is shown).
    private static readonly string QueueName = "Example_Queue";
    private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
    private static IAmazonSQS client;

    public static async Task Main()
    {
        client = new AmazonSQSClient(ServiceRegion);
        var createQueueResponse = await CreateQueue(client, QueueName);

        string queueUrl = createQueueResponse.QueueUrl;
```

```

        Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
    {
        { "Title",    new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
        { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
        { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
    };

    string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";

    var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
messageAttributes);
}

/// <summary>
/// Creates a new Amazon SQS queue using the queue name passed to it
/// in queueName.
/// </summary>
/// <param name="client">An SQS client object used to send the message.</
param>
/// <param name="queueName">A string representing the name of the queue
/// to create.</param>
/// <returns>A CreateQueueResponse that contains information about the
/// newly created queue.</returns>
public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
string queueName)
{
    var request = new CreateQueueRequest
    {
        QueueName = queueName,
        Attributes = new Dictionary<string, string>
        {
            { "DelaySeconds", "60" },
            { "MessageRetentionPeriod", "86400" },
        },
    };

    var response = await client.CreateQueueAsync(request);
    Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");
}

```

```

        return response;
    }

    /// <summary>
    /// Sends a message to an SQS queue.
    /// </summary>
    /// <param name="client">An SQS client object used to send the message.</
param>
    /// <param name="queueUrl">The URL of the queue to which to send the
    /// message.</param>
    /// <param name="messageBody">A string representing the body of the
    /// message to be sent to the queue.</param>
    /// <param name="messageAttributes">Attributes for the message to be
    /// sent to the queue.</param>
    /// <returns>A SendMessageResponse object that contains information
    /// about the message that was sent.</returns>
    public static async Task<SendMessageResponse> SendMessage(
        IAmazonSQS client,
        string queueUrl,
        string messageBody,
        Dictionary<string, MessageAttributeValue> messageAttributes)
    {
        var sendMessageRequest = new SendMessageRequest
        {
            DelaySeconds = 10,
            MessageAttributes = messageAttributes,
            MessageBody = messageBody,
            QueueUrl = queueUrl,
        };

        var response = await client.SendMessageAsync(sendMessageRequest);
        Console.WriteLine($"Sent a message with id : {response.MessageId}");

        return response;
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [SendMessage](#) à la section Référence des AWS SDK pour .NET API.

SetQueueAttributes

L'exemple de code suivant montre comment utiliser `SetQueueAttributes`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Définissez l'attribut de politique d'une file d'attente pour une rubrique.

```

/// <summary>
/// Set the policy attribute of a queue for a topic.
/// </summary>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="queueUrl">The url for the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
{
    var queuePolicy = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +
            "\"Effect\": \"Allow\", " +
            "\"Principal\": { " +
                "\"Service\": " +
                    "\"sns.amazonaws.com\" " +
                "}, " +
            "\"Action\": \"sqs:SendMessage\", " +
            "\"Resource\": \"{queueArn}\", " +
            "\"Condition\": { " +
                "\"ArnEquals\": { " +
                    "\"aws:SourceArn\": \"{topicArn}\" " +
                "}, " +
            "}, " +
        "}] " +
    "};

    var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(

```

```
new SetQueueAttributesRequest()
{
    QueueUrl = queueUrl,
    Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
    });
return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [SetQueueAttributes](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Publier des messages dans des files d'attente

L'exemple de code suivant illustre comment :

- Créer une rubrique (FIFO ou non FIFO).
- Abonner plusieurs files d'attente à la rubrique avec la possibilité d'appliquer un filtre.
- Publier des messages dans la rubrique.
- Interroger les files d'attente pour les messages reçus.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
```

```

private static bool _useFifoTopic = false;
private static bool _useContentBasedDeduplication = false;
private static string _topicName = null!;
private static string _topicArn = null!;

private static readonly int _queueCount = 2;
private static readonly string[] _queueUrls = new string[_queueCount];
private static readonly string[] _subscriptionArns = new string[_queueCount];
private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
public static SNSWrapper SnsWrapper { get; set; } = null!;
public static SQSWrapper SqsWrapper { get; set; } = null!;
public static bool UseConsole { get; set; } = true;
static async Task Main(string[] args)
{
    // Set up dependency injection for Amazon EventBridge.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonSQS>()
                .AddAWSService<IAmazonSimpleNotificationService>()
                .AddTransient<SNSWrapper>()
                .AddTransient<SQSWrapper>()
        )
        .Build();

    ServicesSetup(host);
    PrintDescription();

    await RunScenario();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();

```



```

        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.

```

```

    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
            $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
            $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
            $"\r\nFIFO topics deliver messages in order and support
deduplication and message filtering." +
            $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
"example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
the topic name.\r\n");
        }
    }

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
is supported." +
                        $"\\r\\nDeduplication IDs are either set in the message
or automatically generated " +
                        $"\\r\\nfrom content using a hash function.\\r\\n" +
                        $"\\r\\nIf a message is successfully published to an SNS
FIFO topic, any message " +
                        $"\\r\\npublished and determined to have the same
deduplication ID, " +
                        $"\\r\\nwithin the five-minute deduplication interval,
is accepted but not delivered.\\r\\n" +
                        $"\\r\\nFor more information about deduplication, " +
                        $"\\r\\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

        _useContentBasedDeduplication = GetYesNoResponse("Use content-based
deduplication instead of entering a deduplication ID?");
        Console.WriteLine(new string('-', 80));
    }

    _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
_useContentBasedDeduplication);

    Console.WriteLine($"Your new topic with the name {_topicName}" +
                    $"\\r\\nand Amazon Resource Name (ARN) {_topicArn}" +
                    $"\\r\\nhas been created.\\r\\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)

```

```

        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
                }

                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                $"\r\nand queue URL {queueUrl}" +
                                $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }
    }
}

```

```

    }
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +
                "If you add a filter to this subscription, then only the
filtered messages " +
                "will be received in the queue.");

            Console.WriteLine(
                "For information about message filtering, " +
                "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

            Console.WriteLine(
                "For this example, you can filter messages by a " +
                "TONE attribute.");
        }

        var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

        string? filterPolicy = null;
        if (useFilter)
        {

```

```

        filterPolicy = CreateFilterPolicy();
    }
    var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
    queueArn);
    _subscriptionArns[queueCount] = subscriptionArn;

    Console.WriteLine(
        $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
        $"with the subscription ARN {subscriptionArn}");
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
        int.TryParse(selection, out selectionNumber);
        if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
        {

```

```

        filterSelections.Add(_tones[selectionNumber - 1]);
    }
} while (selectionNumber != 0);

var filters = new Dictionary<string, List<string>>
{
    { "tone", filterSelections }
};
string filterPolicy = JsonSerializer.Serialize(filters);
return filterPolicy;
}

/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                             "\r\nAll messages within the same group will be
received in the order " +
                             "they were published.");

            Console.WriteLine();
            var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

            if (!_useContentBasedDeduplication)
            {

```

```

        Console.WriteLine("Because you are not using content-based
deduplication, " +
                        "you must enter a deduplication ID.");

        Console.WriteLine("Enter a deduplication ID for this message.");
        deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
    }

    if (GetYesNoResponse("Add an attribute to this message?"))
    {
        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageId = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));

```



```

        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                $"{"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
    }

```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
        }
```

```

        Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
{
    if (UseConsole)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);

        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}

/// <summary>
/// Helper method to get a string response from the user through the console.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static string GetUserResponse(string question, string defaultAnswer)
{
    if (UseConsole)
    {
        var response = "";
        while (string.IsNullOrEmpty(response))
        {
            Console.WriteLine(question);
            response = Console.ReadLine();
        }
    }
}

```

```

        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}
}

```

Créez une classe qui englobe les opérations Amazon SQS.

```

/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        }
    }
}

```

```

    };

    var createQueueRequest = new CreateQueueRequest()
    {
        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
        createQueueRequest.Attributes.Add(
            QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

```

```

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
            "\"Version\": \"2012-10-17\", " +
            "\"Statement\": [{" +
                "\"Effect\": \"Allow\", " +
                "\"Principal\": { " +
                    "\"Service\": " +
                        "\"sns.amazonaws.com\"" +
                    "}, " +
                "\"Action\": \"sqs:SendMessage\", " +
                "\"Resource\": \"{queueArn}\", " +
                "\"Condition\": { " +
                    "\"ArnEquals\": { " +
                        "\"aws:SourceArn\": \"{topicArn}\""
+
                    "}" +
                "}" +
            "}] " +
        "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.

```

```

    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

```

```

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}

```

Créez une classe qui englobe les opérations Amazon SNS.

```

/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSNS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
    attributes.

```



```

    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>

```

```

    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,

```

```

        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }

    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>

```

```
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Utiliser le framework de traitement des AWS messages pour .NET avec Amazon SQS

L'exemple de code suivant montre comment créer des applications qui publient et reçoivent des messages Amazon SQS à l'aide de l' AWS infrastructure de traitement des messages pour .NET.

SDK pour .NET

Fournit un didacticiel pour le cadre de traitement des AWS messages pour .NET. Le didacticiel crée une application Web qui permet à l'utilisateur de publier un message Amazon SQS et une application de ligne de commande qui reçoit le message.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez le [didacticiel complet](#) dans le guide du AWS SDK pour .NET développeur et l'exemple ci-dessous [GitHub](#).

Les services utilisés dans cet exemple

- Amazon SQS

Exemples sans serveur

Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une file d'attente SQS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

SDK pour .NET

Note

Il y en a plus à ce sujet [GitHub](#). Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Utilisation d'un événement SQS avec Lambda en utilisant .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
    ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

L'exemple de code suivant montre comment mettre en œuvre une réponse partielle par lots pour les fonctions Lambda qui reçoivent des événements à partir d'une file d'attente SQS. La fonction signale les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'[exemples sans serveur](#).

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer),
    namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

```
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

Exemples de Step Functions utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide de AWS SDK pour .NET with Step Functions.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour Step Functions

L'exemple de code suivant montre comment commencer à utiliser Step Functions.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

public class HelloStepFunctions
{
    static async Task Main()
    {
        var stepFunctionsClient = new AmazonStepFunctionsClient();

        Console.Clear();
        Console.WriteLine("Welcome to AWS Step Functions");
        Console.WriteLine("Let's list up to 10 of your state machines:");
        var stateMachineListRequest = new ListStateMachinesRequest { MaxResults =
10 };

        // Get information for up to 10 Step Functions state machines.
        var response = await
stepFunctionsClient.ListStateMachinesAsync(stateMachineListRequest);

        if (response.StateMachines.Count > 0)
        {
            response.StateMachines.ForEach(stateMachine =>
            {
                Console.WriteLine($"State Machine Name: {stateMachine.Name}\tAmazon
Resource Name (ARN): {stateMachine.StateMachineArn}");
            });
        }
    }
}
```

```
        });  
    }  
    else  
    {  
        Console.WriteLine("\tNo state machines were found.");  
    }  
}  
}
```

- Pour plus de détails sur l'API, reportez-vous [ListStateMachines](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- créer une activité ;
- créer une machine à états à partir d'une définition Amazon States Language qui contient l'activité créée précédemment en tant qu'étape ;
- exécuter la machine d'état et répondre à l'activité saisie par l'utilisateur ;
- obtenir le statut final et le résultat une fois l'exécution terminée, puis nettoyer les ressources.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
global using System.Text.Json;  
global using Amazon.StepFunctions;  
global using Microsoft.Extensions.Configuration;
```

```
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using StepFunctionsActions;
global using LogLevel = Microsoft.Extensions.Logging.LogLevel;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.StepFunctions.Model;

namespace StepFunctionsBasics;

public class StepFunctionsBasics
{
    private static ILogger _logger = null!;
    private static IConfigurationRoot _configuration = null!;
    private static IAmazonIdentityManagementService _iamService = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Step Functions.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonStepFunctions>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<StepFunctionsWrapper>()
                )
            .Build();

        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<StepFunctionsBasics>();

        // Load configuration settings.
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
```

```

        .AddJsonFile("settings.json") // Load test settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

var activityName = _configuration["ActivityName"];
var stateMachineName = _configuration["StateMachineName"];

var roleName = _configuration["RoleName"];
var repoBaseDir = _configuration["RepoBaseDir"];
var jsonFilePath = _configuration["JsonFilePath"];
var jsonFileName = _configuration["JsonFileName"];

var uiMethods = new UiMethods();
var stepFunctionsWrapper =
host.Services.GetRequiredService<StepFunctionsWrapper>();

    _iamService =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();

    // Load definition for the state machine from a JSON file.
    var stateDefinitionJson = File.ReadAllText($"{repoBaseDir}{jsonFilePath}
{jsonFileName}");

    Console.Clear();
    uiMethods.DisplayOverview();
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("Create activity");
    Console.WriteLine("Let's start by creating an activity.");
    string activityArn;
    string stateMachineArn;

    // Check to see if the activity already exists.
    var activityList = await stepFunctionsWrapper.ListActivitiesAsync();
    var existingActivity = activityList.FirstOrDefault(activity => activity.Name
== activityName);
    if (existingActivity is not null)
    {
        activityArn = existingActivity.ActivityArn;
        Console.WriteLine($"Activity, {activityName}, already exists.");
    }
    else
    {

```

```

        activityArn = await stepFunctionsWrapper.CreateActivity(activityName);
    }

    // Swap the placeholder in the JSON file with the Amazon Resource Name (ARN)
    // of the recently created activity.
    var stateDefinition =
stateDefinitionJson.Replace("{{DOC_EXAMPLE_ACTIVITY_ARN}}", activityArn);

    uiMethods.DisplayTitle("Create state machine");
    Console.WriteLine("Now we'll create a state machine.");

    // Find or create an IAM role that can be assumed by Step Functions.
    var role = await GetOrCreateStateMachineRole(roleName);

    // See if the state machine already exists.
    var stateMachineList = await stepFunctionsWrapper.ListStateMachinesAsync();
    var existingStateMachine =
        stateMachineList.FirstOrDefault(stateMachine => stateMachine.Name ==
stateMachineName);
    if (existingStateMachine is not null)
    {
        Console.WriteLine($"State machine, {stateMachineName}, already
exists.");
        stateMachineArn = existingStateMachine.StateMachineArn;
    }
    else
    {
        // Create the state machine.
        stateMachineArn =
            await stepFunctionsWrapper.CreateStateMachine(stateMachineName,
stateDefinition, role.Arn);
        uiMethods.PressEnter();
    }

    Console.WriteLine("The state machine has been created.");
    var describeStateMachineResponse = await
stepFunctionsWrapper.DescribeStateMachineAsync(stateMachineArn);

    Console.WriteLine($"{describeStateMachineResponse.Name}\t{describeStateMachineResponse.Stat
        Console.WriteLine($"Current status: {describeStateMachineResponse.Status}");
        Console.WriteLine($"Amazon Resource Name (ARN) of the role assumed by the
state machine: {describeStateMachineResponse.RoleArn}");

```

```

        var userName = string.Empty;
        Console.WriteLine("Before we start the state machine, tell me what should
ChatSFN call you? ");
        userName = Console.ReadLine();

        // Keep asking until the user enters a string value.
        while (string.IsNullOrEmpty(userName))
        {
            Console.WriteLine("Enter your name: ");
            userName = Console.ReadLine();
        }

        var executionJson = @"{"name": "" + userName + @""}";

        // Start the state machine execution.
        Console.WriteLine("Now we'll start execution of the state machine.");
        var executionArn = await
stepFunctionsWrapper.StartExecutionAsync(executionJson, stateMachineArn);
        Console.WriteLine("State machine started.");

        Console.WriteLine($"Thank you, {userName}. Now let's get started...");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("ChatSFN");

        var isDone = false;
        var response = new GetActivityTaskResponse();
        var taskToken = string.Empty;
        var userChoice = string.Empty;

        while (!isDone)
        {
            response = await stepFunctionsWrapper.GetActivityTaskAsync(activityArn,
"MvpWorker");
            taskToken = response.TaskToken;

            // Parse the returned JSON string.
            var taskJsonResponse = JsonDocument.Parse(response.Input);
            var taskJsonObject = taskJsonResponse.RootElement;
            var message = taskJsonObject.GetProperty("message").GetString();
            var actions =
taskJsonObject.GetProperty("actions").EnumerateArray().Select(x =>
x.ToString()).ToList();
            Console.WriteLine($"\\n{message}\\n");

```

```

        // Prompt the user for another choice.
        Console.WriteLine("ChatSFN: What would you like me to do?");
        actions.ForEach(action => Console.WriteLine($"{action}"));
        Console.Write($"{userName}, tell me your choice: ");
        userChoice = Console.ReadLine();
        if (userChoice?.ToLower() == "done")
        {
            isDone = true;
        }

        Console.WriteLine($"You have selected: {userChoice}");
        var jsonResponse = @"{"action": "" + userChoice + @""}";

        await stepFunctionsWrapper.SendTaskSuccessAsync(taskToken,
jsonResponse);
    }

    await stepFunctionsWrapper.StopExecution(executionArn);
    Console.WriteLine("Now we will wait for the execution to stop.");
    DescribeExecutionResponse executionResponse;
    do
    {
        executionResponse = await
stepFunctionsWrapper.DescribeExecutionAsync(executionArn);
    } while (executionResponse.Status == ExecutionStatus.RUNNING);

    Console.WriteLine("State machine stopped.");
    uiMethods.PressEnter();

    uiMethods.DisplayTitle("State machine executions");
    Console.WriteLine("Now let's take a look at the execution values for the
state machine.");

    // List the executions.
    var executions = await
stepFunctionsWrapper.ListExecutionsAsync(stateMachineArn);

    uiMethods.DisplayTitle("Step function execution values");
    executions.ForEach(execution =>
    {
        Console.WriteLine($"{execution.Name}\t{execution.StartDate} to
{execution.StopDate}");
    });

```

```

        uiMethods.PressEnter();

        // Now delete the state machine and the activity.
        uiMethods.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the state machine...");

        await stepFunctionsWrapper.DeleteStateMachine(stateMachineArn);
        Console.WriteLine("State machine deleted.");

        Console.WriteLine("Deleting the activity...");
        await stepFunctionsWrapper.DeleteActivity(activityArn);
        Console.WriteLine("Activity deleted.");

        Console.WriteLine("The Amazon Step Functions scenario is now complete.");
    }

    static async Task<Role> GetOrCreateStateMachineRole(string roleName)
    {
        // Define the policy document for the role.
        var stateMachineRolePolicy = @"{
            ""Version"": ""2012-10-17"",
            ""Statement"": [{
                ""Sid"": "",
                ""Effect"": ""Allow"",
                ""Principal"": {
                    ""Service"": ""states.amazonaws.com""},
                ""Action"": ""sts:AssumeRole""}]}}";

        var role = new Role();
        var roleExists = false;

        try
        {
            var getRoleResponse = await _iamService.GetRoleAsync(new GetRoleRequest
            { RoleName = roleName });
            roleExists = true;
            role = getRoleResponse.Role;
        }
        catch (NoSuchEntityException)
        {
            // The role doesn't exist. Create it.
            Console.WriteLine($"Role, {roleName} doesn't exist. Creating it...");
        }
    }

```



```

        if (!roleExists)
        {
            var request = new CreateRoleRequest
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = stateMachineRolePolicy,
            };

            var createRoleResponse = await _iamService.CreateRoleAsync(request);
            role = createRoleResponse.Role;
        }

        return role;
    }
}

namespace StepFunctionsBasics;

/// <summary>
/// Some useful methods to make screen display easier.
/// </summary>
public class UiMethods
{
    private readonly string _sepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Welcome to the AWS Step Functions Demo");

        Console.WriteLine("This example application will do the following:");
        Console.WriteLine("\t 1. Create an activity.");
        Console.WriteLine("\t 2. Create a state machine.");
        Console.WriteLine("\t 3. Start an execution.");
        Console.WriteLine("\t 4. Run the worker, then stop it.");
        Console.WriteLine("\t 5. List executions.");
        Console.WriteLine("\t 6. Clean up the resources created for the example.");
    }
}

```

```

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter"></param>
    /// <returns></returns>
    private string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(_sepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(_sepBar);
    }
}

```

Définissez une classe qui encapsule les actions de la machine d'état et des activités.

```

namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

```

```

/// <summary>
/// Wrapper that performs AWS Step Functions actions.
/// </summary>
public class StepFunctionsWrapper
{
    private readonly IAmazonStepFunctions _amazonStepFunctions;

    /// <summary>
    /// The constructor for the StepFunctionsWrapper. Initializes the
    /// client object passed to it.
    /// </summary>
    /// <param name="amazonStepFunctions">An initialized Step Functions client
    object.</param>
    public StepFunctionsWrapper(IAmazonStepFunctions amazonStepFunctions)
    {
        _amazonStepFunctions = amazonStepFunctions;
    }

    /// <summary>
    /// Create a Step Functions activity using the supplied name.
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
    param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
        CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }

    /// <summary>
    /// Create a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
    public async Task<string> CreateStateMachine(string stateMachineName, string
    definition, string roleArn)

```

```

{
    var request = new CreateStateMachineRequest
    {
        Name = stateMachineName,
        Definition = definition,
        RoleArn = roleArn
    };

    var response =
        await _amazonStepFunctions.CreateStateMachineAsync(request);
    return response.StateMachineArn;
}

/// <summary>
/// Delete a Step Machine activity.
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the activity.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteActivity(string activityArn)
{
    var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete a Step Functions state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// state machine.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteStateMachine(string stateMachineArn)
{
    var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
    { StateMachineArn = stateMachineArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>

```

```

    /// Retrieve information about the specified Step Functions execution.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of the
    /// Step Functions execution.</param>
    /// <returns>The API response returned by the API.</returns>
    public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
    executionArn)
    {
        var response = await _amazonStepFunctions.DescribeExecutionAsync(new
    DescribeExecutionRequest { ExecutionArn = executionArn });
        return response;
    }

    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
    returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
    StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
    DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }

    /// <summary>
    /// Retrieve a task with the specified Step Functions activity
    /// with the specified Amazon Resource Name (ARN).
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the Step Functions activity.</param>
    /// <param name="workerName">The name of the Step Functions worker.</param>
    /// <returns>The response from the Step Functions activity.</returns>
    public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
    activityArn, string workerName)
    {
        var response = await _amazonStepFunctions.GetActivityTaskAsync(new
    GetActivityTaskRequest
        { ActivityArn = activityArn, WorkerName = workerName });
    }

```

```

        return response;
    }

    /// <summary>
    /// List the Step Functions activities for the current account.
    /// </summary>
    /// <returns>A list of ActivityListItem.</returns>
    public async Task<List<ActivityListItem>> ListActivitiesAsync()
    {
        var request = new ListActivitiesRequest();
        var activities = new List<ActivityListItem>();

        do
        {
            var response = await _amazonStepFunctions.ListActivitiesAsync(request);

            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }

            activities.AddRange(response.Activities);
        }
        while (request.NextToken is not null);

        return activities;
    }

    /// <summary>
    /// Retrieve information about executions of a Step Functions
    /// state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>A list of ExecutionListItem objects.</returns>
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

```

```

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }

    /// <summary>
    /// Retrieve a list of Step Functions state machines.
    /// </summary>
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }

    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
    /// <param name="taskResponse">The response received from executing the task.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>

```

```

    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
    {
        var executionRequest = new StartExecutionRequest
        {
            Input = executionJson,
            StateMachineArn = stateMachineArn
        };

        var response = await
        _amazonStepFunctions.StartExecutionAsync(executionRequest);
        return response.ExecutionArn;
    }

    /// <summary>
    /// Stop execution of a Step Functions workflow.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of
    /// the Step Functions execution to stop.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StopExecution(string executionArn)
    {
        var response =

```



```
        await _amazonStepFunctions.StopExecutionAsync(new StopExecutionRequest
    { ExecutionArn = executionArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [CreateActivity](#)
 - [CreateStateMachine](#)
 - [DeleteActivity](#)
 - [DeleteStateMachine](#)
 - [DescribeExecution](#)
 - [DescribeStateMachine](#)
 - [GetActivityTask](#)
 - [ListActivities](#)
 - [ListStateMachines](#)
 - [SendTaskSuccess](#)
 - [StartExecution](#)
 - [StopExecution](#)

Actions

CreateActivity

L'exemple de code suivant montre comment utiliser `CreateActivity`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a Step Functions activity using the supplied name.
/// </summary>
/// <param name="activityName">The name for the new Step Functions activity.</
param>
/// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
public async Task<string> CreateActivity(string activityName)
{
    var response = await _amazonStepFunctions.CreateActivityAsync(new
CreateActivityRequest { Name = activityName });
    return response.ActivityArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateActivity](#) à la section Référence des AWS SDK pour .NET API.

CreateStateMachine

L'exemple de code suivant montre comment utiliser `CreateStateMachine`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a Step Functions state machine.
```

```
/// </summary>
/// <param name="stateMachineName">Name for the new Step Functions state
/// machine.</param>
/// <param name="definition">A JSON string that defines the Step Functions
/// state machine.</param>
/// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
/// <returns></returns>
public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
{
    var request = new CreateStateMachineRequest
    {
        Name = stateMachineName,
        Definition = definition,
        RoleArn = roleArn
    };

    var response =
        await _amazonStepFunctions.CreateStateMachineAsync(request);
    return response.StateMachineArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [CreateStateMachine](#) à la section Référence des AWS SDK pour .NET API.

DeleteActivity

L'exemple de code suivant montre comment utiliser `DeleteActivity`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a Step Machine activity.
```

```
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the activity.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteActivity(string activityArn)
{
    var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteActivity](#) à la section Référence des AWS SDK pour .NET API.

DeleteStateMachine

L'exemple de code suivant montre comment utiliser `DeleteStateMachine`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a Step Functions state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// state machine.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteStateMachine(string stateMachineArn)
{
    var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
    { StateMachineArn = stateMachineArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteStateMachine](#) à la section Référence des AWS SDK pour .NET API.

DescribeExecution

L'exemple de code suivant montre comment utiliser `DescribeExecution`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Retrieve information about the specified Step Functions execution.
/// </summary>
/// <param name="executionArn">The Amazon Resource Name (ARN) of the
/// Step Functions execution.</param>
/// <returns>The API response returned by the API.</returns>
public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
{
    var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
    return response;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeExecution](#) à la section Référence des AWS SDK pour .NET API.

DescribeStateMachine

L'exemple de code suivant montre comment utiliser `DescribeStateMachine`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Retrieve information about the specified Step Functions state machine.
/// </summary>
/// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine to retrieve.</param>
/// <returns>Information about the specified Step Functions state machine.</
returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }
```

- Pour plus de détails sur l'API, reportez-vous [DescribeStateMachine](#) à la section Référence des AWS SDK pour .NET API.

GetActivityTask

L'exemple de code suivant montre comment utiliser `GetActivityTask`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Retrieve a task with the specified Step Functions activity
/// with the specified Amazon Resource Name (ARN).
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the Step Functions activity.</param>
/// <param name="workerName">The name of the Step Functions worker.</param>
/// <returns>The response from the Step Functions activity.</returns>
public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
{
    var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
    { ActivityArn = activityArn, WorkerName = workerName });
    return response;
}

```

- Pour plus de détails sur l'API, reportez-vous [GetActivityTask](#) à la section Référence des AWS SDK pour .NET API.

ListActivities

L'exemple de code suivant montre comment utiliser `ListActivities`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// List the Step Functions activities for the current account.
/// </summary>
/// <returns>A list of ActivityListItems.</returns>
public async Task<List<ActivityListItem>> ListActivitiesAsync()
{
    var request = new ListActivitiesRequest();
}

```

```
var activities = new List<ActivityListItem>();

do
{
    var response = await _amazonStepFunctions.ListActivitiesAsync(request);

    if (response.NextToken is not null)
    {
        request.NextToken = response.NextToken;
    }

    activities.AddRange(response.Activities);
}
while (request.NextToken is not null);

return activities;
}
```

- Pour plus de détails sur l'API, reportez-vous [ListActivities](#) à la section Référence des AWS SDK pour .NET API.

ListExecutions

L'exemple de code suivant montre comment utiliser `ListExecutions`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Retrieve information about executions of a Step Functions
/// state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
```



```

    /// <returns>A list of ExecutionListItem objects.</returns>
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }

```

- Pour plus de détails sur l'API, reportez-vous [ListExecutions](#) à la section Référence des AWS SDK pour .NET API.

ListStateMachines

L'exemple de code suivant montre comment utiliser `ListStateMachines`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Retrieve a list of Step Functions state machines.

```

```

    /// </summary>
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }

```

- Pour plus de détails sur l'API, reportez-vous [ListStateMachines](#) à la section Référence des AWS SDK pour .NET API.

SendTaskSuccess

L'exemple de code suivant montre comment utiliser `SendTaskSuccess`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
    /// <param name="taskResponse">The response received from executing the task.</
param>

```

```

    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

```

- Pour plus de détails sur l'API, reportez-vous [SendTaskSuccess](#) à la section Référence des AWS SDK pour .NET API.

StartExecution

L'exemple de code suivant montre comment utiliser `StartExecution`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
    {
        var executionRequest = new StartExecutionRequest

```

```
{
    Input = executionJson,
    StateMachineArn = stateMachineArn
};

var response = await
    _amazonStepFunctions.StartExecutionAsync(executionRequest);
return response.ExecutionArn;
}
```

- Pour plus de détails sur l'API, reportez-vous [StartExecution](#) à la section Référence des AWS SDK pour .NET API.

AWS STS exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with AWS STS.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

AssumeRole

L'exemple de code suivant montre comment utiliser `AssumeRole`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace AssumeRoleExample
{
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
        id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        // A region property may be used if the profile or credentials loaded do not
        specify a region,
        // or to use a specific region.
        private static readonly RegionEndpoint REGION = RegionEndpoint.USSouth1;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of the
            // default user.
```

```

        var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";

        var client = new
Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

        // Get and display the information about the identity of the default
user.
        var callerIdRequest = new GetCallerIdentityRequest();
        var caller = await client.GetCallerIdentityAsync(callerIdRequest);
        Console.WriteLine($"Original Caller: {caller.Arn}");

        // Create the request to use with the AssumeRoleAsync call.
        var assumeRoleReq = new AssumeRoleRequest()
        {
            DurationSeconds = 1600,
            RoleSessionName = "Session1",
            RoleArn = roleArnToAssume
        };

        var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

        // Now create a new client based on the credentials of the caller
assuming the role.
        var client2 = new AmazonSecurityTokenServiceClient(credentials:
assumeRoleRes.Credentials, REGION);

        // Get and display information about the caller that has assumed the
defined role.
        var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
        Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
    }
}
}

```

- Pour plus de détails sur l'API, reportez-vous [AssumeRole](#) à la section Référence des AWS SDK pour .NET API.

Support exemples utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l'aide du AWS SDK pour .NET with Support.

Les principes de base sont des exemples de code qui vous montrent comment effectuer les opérations essentielles au sein d'un service.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Mise en route](#)
- [Principes de base](#)
- [Actions](#)

Mise en route

Bonjour Support

L'exemple de code suivant montre comment démarrer avec Support.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using Amazon.AWSSupport;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public static class HelloSupport
{
    static async Task Main(string[] args)
    {
```

```
// Use the AWS .NET Core Setup package to set up dependency injection for
the AWS Support service.
// Use your AWS profile name, or leave it blank to use the default profile.
// You must have one of the following AWS Support plans: Business,
Enterprise On-Ramp, or Enterprise. Otherwise, an exception will be thrown.
using var host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonAWSSupport>()
    ).Build();

// Now the client is available for injection.
var supportClient = host.Services.GetRequiredService<IAmazonAWSSupport>();

// You can use await and any of the async methods to get a response.
var response = await supportClient.DescribeServicesAsync();
Console.WriteLine($"Hello AWS Support! There are {response.Services.Count}
services available.");
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeServices](#) à la section Référence des AWS SDK pour .NET API.

Principes de base

Principes de base

L'exemple de code suivant illustre comment :

- Obtenez et affichez les services disponibles et les niveaux de gravité des dossiers.
- Créez un dossier de support en utilisant un service, une catégorie et un niveau de gravité sélectionnés.
- Obtenez et affichez une liste des dossiers ouverts pour la journée en cours.
- Ajoutez un ensemble de pièces jointes et une communication au nouveau dossier.
- Décrivez la nouvelle pièce jointe et la nouvelle communication relatives au dossier.
- Résolvez le dossier.
- Obtenez et affichez la liste des dossiers ouverts pour la journée en cours.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

Exécutez un scénario interactif à une invite de commande.

```
/// <summary>
/// Hello AWS Support example.
/// </summary>
public static class SupportCaseScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        To use the AWS Support API, you must have one of the following AWS Support
        plans: Business, Enterprise On-Ramp, or Enterprise.

        This .NET example performs the following tasks:
        1. Get and display services. Select a service from the list.
        2. Select a category from the selected service.
        3. Get and display severity levels and select a severity level from the list.
        4. Create a support case using the selected service, category, and severity
        level.
        5. Get and display a list of open support cases for the current day.
        6. Create an attachment set with a sample text file to add to the case.
        7. Add a communication with the attachment to the support case.
        8. List the communications of the support case.
        9. Describe the attachment set.
        10. Resolve the support case.
        11. Get a list of resolved cases for the current day.
    */

    private static SupportWrapper _supportWrapper = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS Support service.
        // Use your AWS profile name, or leave it blank to use the default profile.
```

```

        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonAWSSupport>(new AWSOptions() { Profile
= "default" })
                    .AddTransient<SupportWrapper>()
            )
            .Build();

var logger = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
}).CreateLogger(typeof(SupportCaseScenario));

_supportWrapper = host.Services.GetRequiredService<SupportWrapper>();

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the AWS Support case example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    var apiSupported = await _supportWrapper.VerifySubscription();
    if (!apiSupported)
    {
        logger.LogError("You must have a Business, Enterprise On-Ramp, or
Enterprise Support " +
                        "plan to use the AWS Support API. \n\tPlease
upgrade your subscription to run these examples.");
        return;
    }

    var service = await DisplayAndSelectServices();

    var category = DisplayAndSelectCategories(service);

    var severityLevel = await DisplayAndSelectSeverity();

    var caseId = await CreateSupportCase(service, category, severityLevel);

```

```

        await DescribeTodayOpenCases();

        var attachmentSetId = await CreateAttachmentSet();

        await AddCommunicationToCase(attachmentSetId, caseId);

        var attachmentId = await ListCommunicationsForCase(caseId);

        await DescribeCaseAttachment(attachmentId);

        await ResolveCase(caseId);

        await DescribeTodayResolvedCases();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("AWS Support case example scenario complete.");
        Console.WriteLine(new string('-', 80));
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
    }
}

/// <summary>
/// List some available services from AWS Support, and select a service for the
example.
/// </summary>
/// <returns>The selected service.</returns>
private static async Task<Service> DisplayAndSelectServices()
{
    Console.WriteLine(new string('-', 80));
    var services = await _supportWrapper.DescribeServices();
    Console.WriteLine($"AWS Support client returned {services.Count}
services.");

    Console.WriteLine($"1. Displaying first 10 services:");
    for (int i = 0; i < 10 && i < services.Count; i++)
    {
        Console.WriteLine($"  {i + 1}. {services[i].Name}");
    }

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > services.Count)

```

```

    {
        Console.WriteLine(
            "Select an example support service by entering a number from the
preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }
    Console.WriteLine(new string('-', 80));

    return services[choiceNumber - 1];
}

/// <summary>
/// List the available categories for a service and select a category for the
example.
/// </summary>
/// <param name="service">Service to use for displaying categories.</param>
/// <returns>The selected category.</returns>
private static Category DisplayAndSelectCategories(Service service)
{
    Console.WriteLine(new string('-', 80));

    Console.WriteLine($"2. Available support categories for Service
\"{service.Name}\":");
    for (int i = 0; i < service.Categories.Count; i++)
    {
        Console.WriteLine($"  {i + 1}. {service.Categories[i].Name}");
    }

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > service.Categories.Count)
    {
        Console.WriteLine(
            "Select an example support category by entering a number from the
preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out choiceNumber);
    }

    Console.WriteLine(new string('-', 80));

    return service.Categories[choiceNumber - 1];
}

```

```

    /// <summary>
    /// List available severity levels from AWS Support, and select a level for the
    example.
    /// </summary>
    /// <returns>The selected severity level.</returns>
    private static async Task<SeverityLevel> DisplayAndSelectSeverity()
    {
        Console.WriteLine(new string('-', 80));
        var severityLevels = await _supportWrapper.DescribeSeverityLevels();

        Console.WriteLine($"3. Get and display available severity levels:");
        for (int i = 0; i < 10 && i < severityLevels.Count; i++)
        {
            Console.WriteLine($"  \t{i + 1}. {severityLevels[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > severityLevels.Count)
        {
            Console.WriteLine(
                "Select an example severity level by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        Console.WriteLine(new string('-', 80));

        return severityLevels[choiceNumber - 1];
    }

    /// <summary>
    /// Create an example support case.
    /// </summary>
    /// <param name="service">Service to use for the new case.</param>
    /// <param name="category">Category to use for the new case.</param>
    /// <param name="severity">Severity to use for the new case.</param>
    /// <returns>The caseId of the new support case.</returns>
    private static async Task<string> CreateSupportCase(Service service,
        Category category, SeverityLevel severity)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Create an example support case" +
            $" with the following settings:" +

```

```

        $" \n\tService: {service.Name}, Category: {category.Name}
" +
        $"and Severity Level: {severity.Name}.";
    var caseId = await _supportWrapper.CreateCase(service.Code, category.Code,
severity.Code,
        "Example case for testing, ignore.", "This is my example support
case.");

    Console.WriteLine($" \tNew case created with ID {caseId}");

    Console.WriteLine(new string('-', 80));

    return caseId;
}

/// <summary>
/// List open cases for the current day.
/// </summary>
/// <returns>Async task.</returns>
private static async Task DescribeTodayOpenCases()
{
    Console.WriteLine($"5. List the open support cases for the current day.");
    // Describe the cases. If it is empty, try again and allow time for the new
case to appear.
    List<CaseDetails> currentOpenCases = null!;
    while (currentOpenCases == null || currentOpenCases.Count == 0)
    {
        Thread.Sleep(1000);
        currentOpenCases = await _supportWrapper.DescribeCases(
            new List<string>(),
            null,
            false,
            false,
            DateTime.UtcNow.Date,
            DateTime.UtcNow);
    }

    foreach (var openCase in currentOpenCases)
    {
        Console.WriteLine($" \tCase: {openCase.CaseId} created
{openCase.TimeCreated}");
    }

    Console.WriteLine(new string('-', 80));
}

```

```

    }

    /// <summary>
    /// Create an attachment set for a support case.
    /// </summary>
    /// <returns>The attachment set id.</returns>
    private static async Task<string> CreateAttachmentSet()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Create an attachment set for a support case.");
        var fileName = "example_attachment.txt";

        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for attachment to a support case.");
        }

        await using var ms = new MemoryStream(await
File.ReadAllBytesAsync(fileName));

        var attachmentSetId = await _supportWrapper.AddAttachmentToSet(
            ms,
            fileName);

        Console.WriteLine($"\\tNew attachment set created with id: \\n
\\t{attachmentSetId.Substring(0, 65)}...");

        Console.WriteLine(new string('-', 80));

        return attachmentSetId;
    }

    /// <summary>
    /// Add an attachment set and communication to a case.
    /// </summary>
    /// <param name="attachmentSetId">Id of the attachment set.</param>
    /// <param name="caseId">Id of the case to receive the attachment set.</param>
    /// <returns>Async task.</returns>
    private static async Task AddCommunicationToCase(string attachmentSetId, string
caseId)
    {

```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Add attachment set and communication to {caseId}.");

        await _supportWrapper.AddCommunicationToCase(
            caseId,
            "This is an example communication added to a support case.",
            attachmentSetId);

        Console.WriteLine($"\\tNew attachment set and communication added to
{caseId}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the communications for a case.
    /// </summary>
    /// <param name="caseId">Id of the case to describe.</param>
    /// <returns>An attachment id.</returns>
    private static async Task<string> ListCommunicationsForCase(string caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. List communications for case {caseId}.");

        var communications = await _supportWrapper.DescribeCommunications(caseId);
        var attachmentId = "";
        foreach (var communication in communications)
        {
            Console.WriteLine(
                $"\\tCommunication created on: {communication.TimeCreated} has
{communication.AttachmentSet.Count} attachments.");
            if (communication.AttachmentSet.Any())
            {
                attachmentId = communication.AttachmentSet.First().AttachmentId;
            }
        }

        Console.WriteLine(new string('-', 80));
        return attachmentId;
    }

    /// <summary>
    /// Describe an attachment by id.
    /// </summary>

```



```

/// <param name="attachmentId">Id of the attachment to describe.</param>
/// <returns>Async task.</returns>
private static async Task DescribeCaseAttachment(string attachmentId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Describe the attachment set.");

    var attachment = await _supportWrapper.DescribeAttachment(attachmentId);
    var data = Encoding.ASCII.GetString(attachment.Data.ToArray());
    Console.WriteLine($"\\tAttachment includes {attachment.FileName} with data:
\\n\\t{data}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Resolve the support case.
/// </summary>
/// <param name="caseId">Id of the case to resolve.</param>
/// <returns>Async task.</returns>
private static async Task ResolveCase(string caseId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"10. Resolve case {caseId}.");

    var status = await _supportWrapper.ResolveCase(caseId);
    Console.WriteLine($"\\tCase {caseId} has final status {status}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List resolved cases for the current day.
/// </summary>
/// <returns>Async Task.</returns>
private static async Task DescribeTodayResolvedCases()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"11. List the resolved support cases for the current
day.");
    var currentCases = await _supportWrapper.DescribeCases(
        new List<string>(),
        null,
        false,

```

```

        true,
        DateTime.UtcNow.Date,
        DateTime.UtcNow);

    foreach (var currentCase in currentCases)
    {
        if (currentCase.Status == "resolved")
        {
            Console.WriteLine(
                $"\\tCase: {currentCase.CaseId}: status {currentCase.Status}");
        }
    }

    Console.WriteLine(new string('-', 80));
}
}

```

Méthodes d'encapsulation utilisées par le scénario pour les Support actions.

```

/// <summary>
/// Wrapper methods to use AWS Support for working with support cases.
/// </summary>
public class SupportWrapper
{
    private readonly IAmazonAWSSupport _amazonSupport;
    public SupportWrapper(IAmazonAWSSupport amazonSupport)
    {
        _amazonSupport = amazonSupport;
    }

    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
    /// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
    are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(

```

```

        new DescribeServicesRequest()
        {
            Language = language
        });
    return response.Services;
}

/// <summary>
/// Get the descriptions of support severity levels.
/// </summary>
/// <param name="name">Optional language for severity levels.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
/// <returns>The list of support severity levels.</returns>
public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
{
    var response = await _amazonSupport.DescribeSeverityLevelsAsync(
        new DescribeSeverityLevelsRequest()
        {
            Language = language
        });
    return response.SeverityLevels;
}

/// <summary>
/// Create a new support case.
/// </summary>
/// <param name="serviceCode">Service code for the new case.</param>
/// <param name="categoryCode">Category for the new case.</param>
/// <param name="severityCode">Severity code for the new case.</param>
/// <param name="subject">Subject of the new case.</param>
/// <param name="body">Body text of the new case.</param>
/// <param name="language">Optional language support for your case.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
/// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>

```

```

    /// <returns>The caseId of the new support case.</returns>
    public async Task<string> CreateCase(string serviceCode, string categoryCode,
    string severityCode, string subject,
        string body, string language = "en", string? attachmentSetId = null, string
    issueType = "customer-service")
    {
        var response = await _amazonSupport.CreateCaseAsync(
            new CreateCaseRequest()
            {
                ServiceCode = serviceCode,
                CategoryCode = categoryCode,
                SeverityCode = severityCode,
                Subject = subject,
                Language = language,
                AttachmentSetId = attachmentSetId,
                IssueType = issueType,
                CommunicationBody = body
            });
        return response.CaseId;
    }

    /// <summary>
    /// Add an attachment to a set, or create a new attachment set if one does not
    exist.
    /// </summary>
    /// <param name="data">The data for the attachment.</param>
    /// <param name="fileName">The file name for the attachment.</param>
    /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
    new attachment set if empty.</param>
    /// <returns>The setId of the attachment.</returns>
    public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
    string? attachmentSetId = null)
    {
        var response = await _amazonSupport.AddAttachmentsToSetAsync(
            new AddAttachmentsToSetRequest
            {
                AttachmentSetId = attachmentSetId,
                Attachments = new List<Attachment>
                {
                    new Attachment
                    {
                        Data = data,

```

```

        FileName = fileName
    }
}

});
return response.AttachmentSetId;
}

/// <summary>
/// Get description of a specific attachment.
/// </summary>
/// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
/// <returns>The attachment object.</returns>
public async Task<Attachment> DescribeAttachment(string attachmentId)
{
    var response = await _amazonSupport.DescribeAttachmentAsync(
        new DescribeAttachmentRequest()
        {
            AttachmentId = attachmentId
        });
    return response.Attachment;
}

/// <summary>
/// Add communication to a case, including optional attachment set ID and CC
email addresses.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <param name="body">Body text of the communication.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set.</param>
/// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
/// <returns>True if successful.</returns>
public async Task<bool> AddCommunicationToCase(string caseId, string body,
    string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
{
    var response = await _amazonSupport.AddCommunicationToCaseAsync(
        new AddCommunicationToCaseRequest()
        {
            CaseId = caseId,
            CommunicationBody = body,

```

```

        AttachmentSetId = attachmentSetId,
        CcEmailAddresses = ccEmailAddresses
    });
    return response.Result;
}

/// <summary>
/// Describe the communications for a case, optionally with a date filter.
/// </summary>
/// <param name="caseId">The ID of the support case.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <returns>The list of communications for the case.</returns>
public async Task<List<Communication>> DescribeCommunications(string caseId,
DateTime? afterTime = null, DateTime? beforeTime = null)
{
    var results = new List<Communication>();
    var paginateCommunications =
_amazonSupport.Paginators.DescribeCommunications(
    new DescribeCommunicationsRequest()
    {
        CaseId = caseId,
        AfterTime = afterTime?.ToString("s"),
        BeforeTime = beforeTime?.ToString("s")
    });
    // Get the entire list using the paginator.
    await foreach (var communications in paginateCommunications.Communications)
    {
        results.Add(communications);
    }
    return results;
}

/// <summary>
/// Get case details for a list of case ids, optionally with date filters.
/// </summary>
/// <param name="caseIds">The list of case IDs.</param>
/// <param name="displayId">Optional display ID.</param>

```

```

    /// <param name="includeCommunication">True to include communication. Defaults
to true.</param>
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
displayId = null, bool includeCommunication = true,
    bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
beforeTime = null,
    string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }

    /// <summary>
    /// Resolve a support case by caseId.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>

```

```
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}

/// <summary>
/// Verify the support level for AWS Support API access.
/// </summary>
/// <returns>True if the subscription level supports API access.</returns>
public async Task<bool> VerifySubscription()
{
    try
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = "en"
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (Amazon.AWSSupport.AmazonAWSSupportException ex)
    {
        if (ex.ErrorCode == "SubscriptionRequiredException")
        {
            return false;
        }
        else throw;
    }
}
}
```

- Pour plus de détails sur l'API, consultez les rubriques suivantes dans la Référence des API du kit AWS SDK pour .NET .
 - [AddAttachmentsToSet](#)

- [AddCommunicationToCase](#)
- [CreateCase](#)
- [DescribeAttachment](#)
- [DescribeCases](#)
- [DescribeCommunications](#)
- [DescribeServices](#)
- [DescribeSeverityLevels](#)
- [ResolveCase](#)

Actions

AddAttachmentsToSet

L'exemple de code suivant montre comment utiliser `AddAttachmentsToSet`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Add an attachment to a set, or create a new attachment set if one does not
exist.
/// </summary>
/// <param name="data">The data for the attachment.</param>
/// <param name="fileName">The file name for the attachment.</param>
/// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
/// <returns>The setId of the attachment.</returns>
public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
{
    var response = await _amazonSupport.AddAttachmentsToSetAsync(
        new AddAttachmentsToSetRequest
```

```

        {
            AttachmentSetId = attachmentSetId,
            Attachments = new List<Attachment>
            {
                new Attachment
                {
                    Data = data,
                    FileName = fileName
                }
            }
        });
    return response.AttachmentSetId;
}

```

- Pour plus de détails sur l'API, reportez-vous [AddAttachmentsToSet](#) à la section Référence des AWS SDK pour .NET API.

AddCommunicationToCase

L'exemple de code suivant montre comment utiliser `AddCommunicationToCase`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Add communication to a case, including optional attachment set ID and CC
email addresses.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <param name="body">Body text of the communication.</param>
/// <param name="attachmentSetId">Optional Id for an attachment set.</param>
/// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
/// <returns>True if successful.</returns>

```

```
public async Task<bool> AddCommunicationToCase(string caseId, string body,
    string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
{
    var response = await _amazonSupport.AddCommunicationToCaseAsync(
        new AddCommunicationToCaseRequest()
        {
            CaseId = caseId,
            CommunicationBody = body,
            AttachmentSetId = attachmentSetId,
            CcEmailAddresses = ccEmailAddresses
        });
    return response.Result;
}
```

- Pour plus de détails sur l'API, reportez-vous [AddCommunicationToCase](#) à la section Référence des AWS SDK pour .NET API.

CreateCase

L'exemple de code suivant montre comment utiliser `CreateCase`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Create a new support case.
/// </summary>
/// <param name="serviceCode">Service code for the new case.</param>
/// <param name="categoryCode">Category for the new case.</param>
/// <param name="severityCode">Severity code for the new case.</param>
/// <param name="subject">Subject of the new case.</param>
/// <param name="body">Body text of the new case.</param>
/// <param name="language">Optional language support for your case.
```

```

    /// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
    are supported.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set for the new
    case.</param>
    /// <param name="issueType">Optional issue type for the new case. Options are
    "customer-service" or "technical".</param>
    /// <returns>The caseId of the new support case.</returns>
    public async Task<string> CreateCase(string serviceCode, string categoryCode,
        string severityCode, string subject,
        string body, string language = "en", string? attachmentSetId = null, string
        issueType = "customer-service")
    {
        var response = await _amazonSupport.CreateCaseAsync(
            new CreateCaseRequest()
            {
                ServiceCode = serviceCode,
                CategoryCode = categoryCode,
                SeverityCode = severityCode,
                Subject = subject,
                Language = language,
                AttachmentSetId = attachmentSetId,
                IssueType = issueType,
                CommunicationBody = body
            });
        return response.CaseId;
    }

```

- Pour plus de détails sur l'API, reportez-vous [CreateCase](#) à la section Référence des AWS SDK pour .NET API.

DescribeAttachment

L'exemple de code suivant montre comment utiliser `DescribeAttachment`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get description of a specific attachment.
/// </summary>
/// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
/// <returns>The attachment object.</returns>
public async Task<Attachment> DescribeAttachment(string attachmentId)
{
    var response = await _amazonSupport.DescribeAttachmentAsync(
        new DescribeAttachmentRequest()
        {
            AttachmentId = attachmentId
        });
    return response.Attachment;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeAttachment](#) à la section Référence des AWS SDK pour .NET API.

DescribeCases

L'exemple de code suivant montre comment utiliser `DescribeCases`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Get case details for a list of case ids, optionally with date filters.
    /// </summary>
    /// <param name="caseIds">The list of case IDs.</param>
    /// <param name="displayId">Optional display ID.</param>
    /// <param name="includeCommunication">True to include communication. Defaults
to true.</param>
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
displayId = null, bool includeCommunication = true,
        bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
beforeTime = null,
        string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }

```

- Pour plus de détails sur l'API, reportez-vous [DescribeCases](#) à la section Référence des AWS SDK pour .NET API.

DescribeCommunications

L'exemple de code suivant montre comment utiliser `DescribeCommunications`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Describe the communications for a case, optionally with a date filter.
/// </summary>
/// <param name="caseId">The ID of the support case.</param>
/// <param name="afterTime">The optional start date for a filtered search.</
param>
/// <param name="beforeTime">The optional end date for a filtered search.</
param>
/// <returns>The list of communications for the case.</returns>
public async Task<List<Communication>> DescribeCommunications(string caseId,
DateTime? afterTime = null, DateTime? beforeTime = null)
{
    var results = new List<Communication>();
    var paginateCommunications =
_amazonSupport.Paginators.DescribeCommunications(
    new DescribeCommunicationsRequest()
    {
        CaseId = caseId,
        AfterTime = afterTime?.ToString("s"),
        BeforeTime = beforeTime?.ToString("s")
    });
    // Get the entire list using the paginator.
```

```
await foreach (var communications in paginateCommunications.Communications)
{
    results.Add(communications);
}
return results;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeCommunications](#) à la section Référence des AWS SDK pour .NET API.

DescribeServices

L'exemple de code suivant montre comment utiliser `DescribeServices`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the descriptions of AWS services.
/// </summary>
/// <param name="name">Optional language for services.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
/// <returns>The list of AWS service descriptions.</returns>
public async Task<List<Service>> DescribeServices(string language = "en")
{
    var response = await _amazonSupport.DescribeServicesAsync(
        new DescribeServicesRequest()
        {
            Language = language
        });
    return response.Services;
}
```


- Pour plus de détails sur l'API, reportez-vous [DescribeServices](#) à la section Référence des AWS SDK pour .NET API.

DescribeSeverityLevels

L'exemple de code suivant montre comment utiliser `DescribeSeverityLevels`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get the descriptions of support severity levels.
/// </summary>
/// <param name="name">Optional language for severity levels.
/// Currently Chinese ("zh"), English ("en"), Japanese ("ja") and Korean ("ko")
are supported.</param>
/// <returns>The list of support severity levels.</returns>
public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
{
    var response = await _amazonSupport.DescribeSeverityLevelsAsync(
        new DescribeSeverityLevelsRequest()
        {
            Language = language
        });
    return response.SeverityLevels;
}
```

- Pour plus de détails sur l'API, reportez-vous [DescribeSeverityLevels](#) à la section Référence des AWS SDK pour .NET API.

ResolveCase

L'exemple de code suivant montre comment utiliser `ResolveCase`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Resolve a support case by caseId.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}
```

- Pour plus de détails sur l'API, reportez-vous [ResolveCase](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon Textract utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Textract.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Scénarios](#)

Scénarios

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda

- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemples d'Amazon Transcribe utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Transcribe.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)

Actions

CreateVocabulary

L'exemple de code suivant montre comment utiliser CreateVocabulary.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>  
/// Create a custom vocabulary using a list of phrases. Custom vocabularies  
/// improve transcription accuracy for one or more specific words.
```

```

    /// </summary>
    /// <param name="languageCode">The language code of the vocabulary.</param>
    /// <param name="phrases">Phrases to use in the vocabulary.</param>
    /// <param name="vocabularyName">Name for the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> CreateCustomVocabulary(LanguageCode
languageCode,
        List<string> phrases, string vocabularyName)
    {
        var response = await _amazonTranscribeService.CreateVocabularyAsync(
            new CreateVocabularyRequest
            {
                LanguageCode = languageCode,
                Phrases = phrases,
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }

```

- Pour plus de détails sur l'API, reportez-vous [CreateVocabulary](#) à la section Référence des AWS SDK pour .NET API.

DeleteMedicalTranscriptionJob

L'exemple de code suivant montre comment utiliser `DeleteMedicalTranscriptionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Delete a medical transcription job. Also deletes the transcript associated
    with the job.
    /// </summary>

```

```
/// <param name="jobName">Name of the medical transcription job to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteMedicalTranscriptionJob(string jobName)
{
    var response = await
        _amazonTranscribeService.DeleteMedicalTranscriptionJobAsync(
            new DeleteMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName
            });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteMedicalTranscriptionJob](#) à la section Référence des AWS SDK pour .NET API.

DeleteTranscriptionJob

L'exemple de code suivant montre comment utiliser `DeleteTranscriptionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete a transcription job. Also deletes the transcript associated with the
job.
/// </summary>
/// <param name="jobName">Name of the transcription job to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTranscriptionJob(string jobName)
{
    var response = await _amazonTranscribeService.DeleteTranscriptionJobAsync(
```

```
        new DeleteTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteTranscriptionJob](#) à la section Référence des AWS SDK pour .NET API.

DeleteVocabulary

L'exemple de code suivant montre comment utiliser `DeleteVocabulary`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Delete an existing custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.DeleteVocabularyAsync(
        new DeleteVocabularyRequest
        {
            VocabularyName = vocabularyName
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Pour plus de détails sur l'API, reportez-vous [DeleteVocabulary](#) à la section Référence des AWS SDK pour .NET API.

GetTranscriptionJob

L'exemple de code suivant montre comment utiliser `GetTranscriptionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get details about a transcription job.
/// </summary>
/// <param name="jobName">A unique name for the transcription job.</param>
/// <returns>A TranscriptionJob instance with information on the requested
job.</returns>
public async Task<TranscriptionJob> GetTranscriptionJob(string jobName)
{
    var response = await _amazonTranscribeService.GetTranscriptionJobAsync(
        new GetTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName
        });
    return response.TranscriptionJob;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetTranscriptionJob](#) à la section Référence des AWS SDK pour .NET API.

GetVocabulary

L'exemple de code suivant montre comment utiliser `GetVocabulary`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Get information about a custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary.</param>
/// <returns>The state of the custom vocabulary.</returns>
public async Task<VocabularyState> GetCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.GetVocabularyAsync(
        new GetVocabularyRequest()
        {
            VocabularyName = vocabularyName
        });
    return response.VocabularyState;
}
```

- Pour plus de détails sur l'API, reportez-vous [GetVocabulary](#) à la section Référence des AWS SDK pour .NET API.

ListMedicalTranscriptionJobs

L'exemple de code suivant montre comment utiliser `ListMedicalTranscriptionJobs`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// List medical transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the medical
transcription jobs.</param>
    /// <returns>A list of summaries about medical transcription jobs.</returns>
    public async Task<List<MedicalTranscriptionJobSummary>>
ListMedicalTranscriptionJobs(
    string? jobNameContains = null)
    {
        var response = await
_amazonTranscribeService.ListMedicalTranscriptionJobsAsync(
    new ListMedicalTranscriptionJobsRequest()
    {
        JobNameContains = jobNameContains
    });
        return response.MedicalTranscriptionJobSummaries;
    }

```

- Pour plus de détails sur l'API, reportez-vous [ListMedicalTranscriptionJobs](#) à la section Référence des AWS SDK pour .NET API.

ListTranscriptionJobs

L'exemple de code suivant montre comment utiliser `ListTranscriptionJobs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// List transcription jobs, optionally with a name filter.
    /// </summary>

```

```

    /// <param name="jobNameContains">Optional name filter for the transcription
    jobs.</param>
    /// <returns>A list of transcription job summaries.</returns>
    public async Task<List<TranscriptionJobSummary>> ListTranscriptionJobs(string?
    jobNameContains = null)
    {
        var response = await _amazonTranscribeService.ListTranscriptionJobsAsync(
            new ListTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.TranscriptionJobSummaries;
    }

```

- Pour plus de détails sur l'API, reportez-vous [ListTranscriptionJobs](#) à la section Référence des AWS SDK pour .NET API.

ListVocabularies

L'exemple de code suivant montre comment utiliser `ListVocabularies`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// List custom vocabularies for the current account. Optionally specify a name
    /// filter and a specific state to filter the vocabularies list.
    /// </summary>
    /// <param name="nameContains">Optional string the vocabulary name must
    contain.</param>
    /// <param name="stateEquals">Optional state of the vocabulary.</param>
    /// <returns>List of information about the vocabularies.</returns>

```

```

public async Task<List<VocabularyInfo>> ListCustomVocabularies(string?
nameContains = null,
    VocabularyState? stateEquals = null)
{
    var response = await _amazonTranscribeService.ListVocabulariesAsync(
        new ListVocabulariesRequest()
        {
            NameContains = nameContains,
            StateEquals = stateEquals
        });
    return response.Vocabularies;
}

```

- Pour plus de détails sur l'API, reportez-vous [ListVocabularies](#) à la section Référence des AWS SDK pour .NET API.

StartMedicalTranscriptionJob

L'exemple de code suivant montre comment utiliser `StartMedicalTranscriptionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

/// <summary>
/// Start a medical transcription job for a media file. This method returns
/// as soon as the job is started.
/// </summary>
/// <param name="jobName">A unique name for the medical transcription job.</
param>
/// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
location.</param>
/// <param name="mediaFormat">The format of the media file.</param>

```

```

    /// <param name="outputBucketName">Location for the output, typically an Amazon
    S3 location.</param>
    /// <param name="transcriptionType">Conversation or dictation transcription
    type.</param>
    /// <returns>A MedicalTransactionJob instance with information on the new job.</
returns>
    public async Task<MedicalTranscriptionJob> StartMedicalTranscriptionJob(
        string jobName, string mediaFileUri,
        MediaFormat mediaFormat, string outputBucketName,
        Amazon.TranscribeService.Type transcriptionType)
    {
        var response = await
        _amazonTranscribeService.StartMedicalTranscriptionJobAsync(
            new StartMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
                LanguageCode =
                    LanguageCode
                        .EnUS, // The value must be en-US for medical
transcriptions.
                OutputBucketName = outputBucketName,
                OutputKey =
                    jobName, // The value is a key used to fetch the output of the
transcription.
                Specialty = Specialty.PRIMARYCARE, // The value PRIMARYCARE must be
set.
                Type = transcriptionType
            });
        return response.MedicalTranscriptionJob;
    }

```

- Pour plus de détails sur l'API, reportez-vous [StartMedicalTranscriptionJob](#) à la section Référence des AWS SDK pour .NET API.

StartTranscriptionJob

L'exemple de code suivant montre comment utiliser `StartTranscriptionJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
/// <summary>
/// Start a transcription job for a media file. This method returns
/// as soon as the job is started.
/// </summary>
/// <param name="jobName">A unique name for the transcription job.</param>
/// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
location.</param>
/// <param name="mediaFormat">The format of the media file.</param>
/// <param name="languageCode">The language code of the media file, such as en-
US.</param>
/// <param name="vocabularyName">Optional name of a custom vocabulary.</param>
/// <returns>A TranscriptionJob instance with information on the new job.</
returns>
public async Task<TranscriptionJob> StartTranscriptionJob(string jobName, string
mediaFileUri,
    MediaFormat mediaFormat, LanguageCode languageCode, string? vocabularyName)
{
    var response = await _amazonTranscribeService.StartTranscriptionJobAsync(
        new StartTranscriptionJobRequest()
        {
            TranscriptionJobName = jobName,
            Media = new Media()
            {
                MediaFileUri = mediaFileUri
            },
            MediaFormat = mediaFormat,
            LanguageCode = languageCode,
            Settings = vocabularyName != null ? new Settings()
            {
                VocabularyName = vocabularyName
            }
        }
    );
}
```

```

        } : null
    });
    return response.TranscriptionJob;
}

```

- Pour plus de détails sur l'API, reportez-vous [StartTranscriptionJob](#) à la section Référence des AWS SDK pour .NET API.

UpdateVocabulary

L'exemple de code suivant montre comment utiliser `UpdateVocabulary`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```

    /// <summary>
    /// Update a custom vocabulary with new values. Update overwrites all existing
    information.
    /// </summary>
    /// <param name="languageCode">The language code of the vocabulary.</param>
    /// <param name="phrases">Phrases to use in the vocabulary.</param>
    /// <param name="vocabularyName">Name for the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> UpdateCustomVocabulary(LanguageCode
    languageCode,
        List<string> phrases, string vocabularyName)
    {
        var response = await _amazonTranscribeService.UpdateVocabularyAsync(
            new UpdateVocabularyRequest()
            {
                LanguageCode = languageCode,
                Phrases = phrases,
                VocabularyName = vocabularyName
            }
        );
    }

```

```
    });  
    return response.VocabularyState;  
}
```

- Pour plus de détails sur l'API, reportez-vous [UpdateVocabulary](#) à la section Référence des AWS SDK pour .NET API.

Exemples d'Amazon Translate utilisant SDK pour .NET

Les exemples de code suivants vous montrent comment effectuer des actions et implémenter des scénarios courants à l' AWS SDK pour .NET aide d'Amazon Translate.

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code en contexte.

Rubriques

- [Actions](#)
- [Scénarios](#)

Actions

DescribeTextTranslationJob

L'exemple de code suivant montre comment utiliser `DescribeTextTranslationJob`.

SDK pour .NET

 Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// The following example shows how to retrieve the details of
/// a text translation job using Amazon Translate.
/// </summary>
public class DescribeTextTranslation
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();

        // The Job Id is generated when the text translation job is started
        // with a call to the StartTextTranslationJob method.
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new DescribeTextTranslationJobRequest
        {
            JobId = jobId,
        };

        var jobProperties = await DescribeTranslationJobAsync(client, request);

        DisplayTranslationJobDetails(jobProperties);
    }

    /// <summary>
    /// Retrieve information about an Amazon Translate text translation job.
    /// </summary>
    /// <param name="client">The initialized Amazon Translate client object.</
param>
```

```

    /// <param name="request">The DescribeTextTranslationJobRequest object.</
param>
    /// <returns>The TextTranslationJobProperties object containing
    /// information about the text translation job..</returns>
    public static async Task<TextTranslationJobProperties>
DescribeTranslationJobAsync(
    AmazonTranslateClient client,
    DescribeTextTranslationJobRequest request)
    {
        var response = await client.DescribeTextTranslationJobAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            return response.TextTranslationJobProperties;
        }
        else
        {
            return null;
        }
    }

    /// <summary>
    /// Displays the properties of the text translation job.
    /// </summary>
    /// <param name="jobProperties">The properties of the text translation
    /// job returned by the call to DescribeTextTranslationJobAsync.</param>
    public static void DisplayTranslationJobDetails(TextTranslationJobProperties
jobProperties)
    {
        if (jobProperties is null)
        {
            Console.WriteLine("No text translation job properties found.");
            return;
        }

        // Display the details of the text translation job.
        Console.WriteLine($"{jobProperties.JobId}: {jobProperties.JobName}");
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [DescribeTextTranslationJob](#) à la section Référence des AWS SDK pour .NET API.

ListTextTranslationJobs

L'exemple de code suivant montre comment utiliser `ListTextTranslationJobs`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// List Amazon Translate translation jobs, along with details about each job.
/// </summary>
public class ListTranslationJobs
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var filter = new TextTranslationJobFilter
        {
            JobStatus = "COMPLETED",
        };

        var request = new ListTextTranslationJobsRequest
        {
            MaxResults = 10,
            Filter = filter,
        };

        await ListJobsAsync(client, request);
    }

    /// <summary>
    /// List Amazon Translate text translation jobs.
    /// </summary>
```

```

    /// <param name="client">The initialized Amazon Translate client object.</
param>
    /// <param name="request">An Amazon Translate
    /// ListTextTranslationJobsRequest object detailing which text
    /// translation jobs are of interest.</param>
    public static async Task ListJobsAsync(
        AmazonTranslateClient client,
        ListTextTranslationJobsRequest request)
    {
        ListTextTranslationJobsResponse response;

        do
        {
            response = await client.ListTextTranslationJobsAsync(request);

            ShowTranslationJobDetails(response.TextTranslationJobPropertiesList);

            request.NextToken = response.NextToken;
        }
        while (response.NextToken is not null);
    }

    /// <summary>
    /// List existing translation job details.
    /// </summary>
    /// <param name="properties">A list of Amazon Translate text
    /// translation jobs.</param>
    public static void
    ShowTranslationJobDetails(List<TextTranslationJobProperties> properties)
    {
        properties.ForEach(prop =>
        {
            Console.WriteLine($"{prop.JobId}: {prop.JobName}");
            Console.WriteLine($"Status: {prop.JobStatus}");
            Console.WriteLine($"Submitted time: {prop.SubmittedTime}");
        });
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [ListTextTranslationJobs](#) à la section Référence des AWS SDK pour .NET API.

StartTextTranslationJob

L'exemple de code suivant montre comment utiliser `StartTextTranslationJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// This example shows how to use Amazon Translate to process the files in
/// an Amazon Simple Storage Service (Amazon S3) bucket. The translated results
/// will also be stored in an Amazon S3 bucket.
/// </summary>
public class BatchTranslate
{
    public static async Task Main()
    {
        var contentType = "text/plain";

        // Set this variable to an S3 bucket location with a folder."
        // Input files must be in a folder and not at the bucket root."
        var s3InputUri = "s3://amzn-s3-demo-bucket1/FOLDER/";
        var s3OutputUri = "s3://amzn-s3-demo-bucket2/";

        // This role must have permissions to read the source bucket and to read
and
        // write to the destination bucket where the translated text will be
stored.
        var dataAccessRoleArn = "arn:aws:iam::0123456789ab:role/
S3TranslateRole";

        var client = new AmazonTranslateClient();
```

```

        var inputConfig = new InputDataConfig
        {
            ContentType = contentType,
            S3Uri = s3InputUri,
        };

        var outputConfig = new OutputDataConfig
        {
            S3Uri = s3OutputUri,
        };

        var request = new StartTextTranslationJobRequest
        {
            JobName = "ExampleTranslationJob",
            DataAccessRoleArn = dataAccessRoleArn,
            InputDataConfig = inputConfig,
            OutputDataConfig = outputConfig,
            SourceLanguageCode = "en",
            TargetLanguageCodes = new List<string> { "fr" },
        };

        var response = await StartTextTranslationAsync(client, request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{response.JobId}: {response.JobStatus}");
        }
    }

    /// <summary>
    /// Start the Amazon Translate text translation job.
    /// </summary>
    /// <param name="client">The initialized AmazonTranslateClient object.</
param>
    /// <param name="request">The request object that includes details such
    /// as source and destination bucket names and the IAM Role that will
    /// be used to access the buckets.</param>
    /// <returns>The StartTextTranslationResponse object that includes the
    /// details of the request response.</returns>
    public static async Task<StartTextTranslationJobResponse>
    StartTextTranslationAsync(AmazonTranslateClient client,
    StartTextTranslationJobRequest request)
    {
        var response = await client.StartTextTranslationJobAsync(request);
    }

```

```
        return response;
    }
}
```

- Pour plus de détails sur l'API, reportez-vous [StartTextTranslationJob](#) à la section Référence des AWS SDK pour .NET API.

StopTextTranslationJob

L'exemple de code suivant montre comment utiliser `StopTextTranslationJob`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// Shows how to stop a running Amazon Translation Service text translation
/// job.
/// </summary>
public class StopTextTranslationJob
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new StopTextTranslationJobRequest
        {
            JobId = jobId,
        };
    }
}
```

```

        await StopTranslationJobAsync(client, request);
    }

    /// <summary>
    /// Sends a request to stop a text translation job.
    /// </summary>
    /// <param name="client">Initialized AmazonTrnslateClient object.</param>
    /// <param name="request">The request object to be passed to the
    /// StopTextJobAsync method.</param>
    public static async Task StopTranslationJobAsync(
        AmazonTranslateClient client,
        StopTextTranslationJobRequest request)
    {
        var response = await client.StopTextTranslationJobAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{response.JobId} as status:
{response.JobStatus}");
        }
    }
}

```

- Pour plus de détails sur l'API, reportez-vous [StopTextTranslationJob](#) à la section Référence des AWS SDK pour .NET API.

TranslateText

L'exemple de code suivant montre comment utiliser `TranslateText`.

SDK pour .NET

Note

Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le [référentiel d'exemples de code AWS](#).

```
using System;
```



```
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// Take text from a file stored a Amazon Simple Storage Service (Amazon S3)
/// object and translate it using the Amazon Transfer Service.
/// </summary>
public class TranslateText
{
    public static async Task Main()
    {
        // If the region you want to use is different from the region
        // defined for the default user, supply it as a parameter to the
        // Amazon Translate client object constructor.
        var client = new AmazonTranslateClient();

        // Set the source language to "auto" to request Amazon Translate to
        // automatically detect te language of the source text.

        // You can get a list of the languages supposed by Amazon Translate
        // in the Amazon Translate Developer's Guide here:
        //      https://docs.aws.amazon.com/translate/latest/dg/what-is.html
        string srcLang = "en"; // English.
        string destLang = "fr"; // French.

        // The Amazon Simple Storage Service (Amazon S3) bucket where the
        // source text file is stored.
        string srcBucket = "amzn-s3-demo-bucket";
        string srcTextFile = "source.txt";

        var srcText = await GetSourceTextAsync(srcBucket, srcTextFile);
        var destText = await TranslatingTextAsync(client, srcLang, destLang,
srcText);

        ShowText(srcText, destText);
    }

    /// <summary>
    /// Use the Amazon S3 TransferUtility to retrieve the text to translate
    /// from an object in an S3 bucket.
```

```

    /// </summary>
    /// <param name="srcBucket">The name of the S3 bucket where the
    /// text is stored.
    /// </param>
    /// <param name="srcTextFile">The key of the S3 object that
    /// contains the text to translate.</param>
    /// <returns>A string representing the source text.</returns>
    public static async Task<string> GetSourceTextAsync(string srcBucket, string
srcTextFile)
    {
        string srcText = string.Empty;

        var s3Client = new AmazonS3Client();
        TransferUtility utility = new TransferUtility(s3Client);

        using var stream = await utility.OpenStreamAsync(srcBucket,
srcTextFile);

        StreamReader file = new System.IO.StreamReader(stream);

        srcText = file.ReadToEnd();
        return srcText;
    }

    /// <summary>
    /// Use the Amazon Translate Service to translate the document from the
    /// source language to the specified destination language.
    /// </summary>
    /// <param name="client">The Amazon Translate Service client used to
    /// perform the translation.</param>
    /// <param name="srcLang">The language of the source text.</param>
    /// <param name="destLang">The destination language for the translated
    /// text.</param>
    /// <param name="text">A string representing the text to ranslate.</param>
    /// <returns>The text that has been translated to the destination
    /// language.</returns>
    public static async Task<string> TranslatingTextAsync(AmazonTranslateClient
client, string srcLang, string destLang, string text)
    {
        var request = new TranslateTextRequest
        {
            SourceLanguageCode = srcLang,
            TargetLanguageCode = destLang,
            Text = text,

```

```
};

var response = await client.TranslateTextAsync(request);

return response.TranslatedText;
}

/// <summary>
/// Show the original text followed by the translated text.
/// </summary>
/// <param name="srcText">The original text to be translated.</param>
/// <param name="destText">The translated text.</param>
public static void ShowText(string srcText, string destText)
{
    Console.WriteLine("Source text:");
    Console.WriteLine(srcText);
    Console.WriteLine();
    Console.WriteLine("Translated text:");
    Console.WriteLine(destText);
}
}
```

- Pour plus de détails sur l'API, reportez-vous [TranslateText](#) à la section Référence des AWS SDK pour .NET API.

Scénarios

Création d'une application Amazon SNS

L'exemple de code suivant montre comment créer une application dotée de fonctionnalités d'abonnement et de publication et qui traduit les messages.

SDK pour .NET

Indique comment utiliser l'API .NET Amazon Simple Notification Service pour créer une application Web dotée de fonctionnalités d'abonnement et de publication. De plus, cet exemple d'application traduit également des messages.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur [GitHub](#).

Les services utilisés dans cet exemple

- Amazon SNS
- Amazon Translate

Créez une application pour analyser les commentaires des clients

L'exemple de code suivant montre comment créer une application qui analyse les cartes de commentaires des clients, les traduit depuis leur langue d'origine, détermine leur sentiment et génère un fichier audio à partir du texte traduit.

SDK pour .NET

Cet exemple d'application analyse et stocke les cartes de commentaires des clients. Plus précisément, elle répond aux besoins d'un hôtel fictif situé à New York. L'hôtel reçoit les commentaires des clients dans différentes langues sous la forme de cartes de commentaires physiques. Ces commentaires sont chargés dans l'application via un client Web. Après avoir chargé l'image d'une carte de commentaires, les étapes suivantes se déroulent :

- Le texte est extrait de l'image à l'aide d'Amazon Textract.
- Amazon Comprehend détermine le sentiment du texte extrait et sa langue.
- Le texte extrait est traduit en anglais à l'aide d'Amazon Translate.
- Amazon Polly synthétise un fichier audio à partir du texte extrait.

L'application complète peut être déployée avec AWS CDK. Pour le code source et les instructions de déploiement, consultez le projet dans [GitHub](#).

Les services utilisés dans cet exemple

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Sécurité de ce AWS produit ou service

Chez Amazon Web Services (AWS), la sécurité dans le cloud est la priorité principale. En tant que client AWS, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses sur la sécurité. La sécurité est une responsabilité partagée entre vous AWS et vous. Le [modèle de responsabilité partagée](#) décrit cela comme la sécurité du cloud et la sécurité dans le cloud.

Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute tous les services proposés dans le AWS cloud et de vous fournir des services que vous pouvez utiliser en toute sécurité. Notre responsabilité en matière de sécurité est notre priorité absolue AWS, et l'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des [programmes de AWS conformité](#).

Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez et par d'autres facteurs, notamment la sensibilité de vos données, les exigences de votre organisation et les lois et réglementations applicables.

Ce AWS produit ou service suit le [modèle de responsabilité partagée](#) par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la [AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité](#).

Rubriques

- [Protection des données dans ce AWS produit ou service](#)
- [Gestion de l'identité et des accès](#)
- [Validation de conformité pour ce AWS produit ou service](#)
- [Résilience pour ce AWS produit ou service](#)
- [Sécurité de l'infrastructure pour ce AWS produit ou service](#)
- [Appliquer une version minimale de TLS dans le AWS SDK pour .NET](#)
- [Migration du client de chiffrement Amazon S3 \(V1 vers V2\)](#)
- [Migration du client de chiffrement Amazon S3 \(V2 vers V4\)](#)

Protection des données dans ce AWS produit ou service

Le [modèle de responsabilité AWS partagée](#) de s'applique à la protection des données dans ce AWS produit ou service. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez [Questions fréquentes \(FAQ\) sur la confidentialité des données](#). Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog [Modèle de responsabilité partagée d'AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section [Utilisation des CloudTrail sentiers](#) dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez [Norme FIPS \(Federal Information Processing Standard\) 140-3](#).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec ce AWS produit ou service ou un autre à

Services AWS l'aide de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Gestion de l'identité et des accès

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS les ressources. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- [Public ciblé](#)
- [Authentification par des identités](#)
- [Gestion de l'accès à l'aide de politiques](#)
- [Comment Services AWS travailler avec IAM](#)
- [Résolution des problèmes AWS d'identité et d'accès](#)

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction du travail que vous effectuez. AWS

Utilisateur du service : si vous avez l' Services AWS habitude de faire votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de nouvelles AWS fonctionnalités pour effectuer votre travail, vous aurez peut-être besoin d'autorisations supplémentaires. Si vous comprenez bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans AWS, consultez [Résolution des problèmes AWS d'identité et d'accès](#) le guide de l'utilisateur du Service AWS que vous utilisez.

Administrateur du service — Si vous êtes responsable des AWS ressources de votre entreprise, vous avez probablement un accès complet à AWS. C'est à vous de déterminer les AWS fonctionnalités et les ressources auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs

de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec AWS, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Administrateur IAM – Si vous êtes un administrateur IAM, vous souhaitez peut-être en savoir plus sur la façon d'écrire des politiques pour gérer l'accès à AWS. Pour consulter des exemples de politiques AWS basées sur l'identité que vous pouvez utiliser dans IAM, consultez le guide de l'utilisateur Service AWS que vous utilisez.

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez [Connexion à votre Compte AWS](#) dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez [Signature AWS Version 4 pour les demandes d'API](#) dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez [Tâches qui requièrent les informations d'identification de l'utilisateur racine](#) dans le Guide de l'utilisateur IAM.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations

d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez [Qu'est-ce que IAM Identity Center ?](#) dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir [Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires](#) dans le guide de l'utilisateur IAM.

[Les groupes IAM](#) spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez [Cas d'utilisation pour les utilisateurs IAM](#) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un [rôle IAM](#) est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en [passant d'un rôle d'utilisateur à un rôle IAM \(console\)](#) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez [Méthodes pour endosser un rôle](#) dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez [Vue d'ensemble des politiques JSON](#) dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Définition d'autorisations IAM personnalisées avec des politiques gérées par le client](#) dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez [Choix entre les politiques gérées et les politiques en ligne](#) dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Vous devez [spécifier un principal](#) dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la [présentation de la liste de contrôle d'accès \(ACL\)](#) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez [Limites d'autorisations pour des entités IAM](#) dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez [Politiques de contrôle de service](#) dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir [Politiques de contrôle des ressources \(RCPs\)](#) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez [Politiques de session](#) dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section [Logique d'évaluation des politiques](#) dans le guide de l'utilisateur IAM.

Comment Services AWS travailler avec IAM

Pour obtenir une vue d'ensemble du Services AWS fonctionnement de la plupart des fonctionnalités IAM, consultez les [AWS services compatibles avec IAM](#) dans le guide de l'utilisateur IAM.

Pour savoir comment utiliser un service spécifique Service AWS avec IAM, consultez la section relative à la sécurité du guide de l'utilisateur du service concerné.

Résolution des problèmes AWS d'identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IAM.

Rubriques

- [Je ne suis pas autorisé à effectuer une action dans AWS](#)
- [Je ne suis pas autorisé à effectuer iam : PassRole](#)
- [Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources](#)

Je ne suis pas autorisé à effectuer une action dans AWS

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource `my-example-widget` fictive, mais ne dispose pas des autorisations `aws:GetWidget` fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource `my-example-widget` à l'aide de l'action `aws:GetWidget`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter `iam:PassRole` l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à AWS.

Certains services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans AWS. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS ressources

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si ces fonctionnalités sont prises AWS en charge, consultez [Comment Services AWS travailler avec IAM](#).
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section [Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez](#) dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section [Fournir un accès à des ressources Comptes AWS détenues par des tiers](#) dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez [Fournir un accès à des utilisateurs authentifiés en externe \(fédération d'identité\)](#) dans le Guide de l'utilisateur IAM.

- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Validation de conformité pour ce AWS produit ou service

Pour savoir si un [programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité](#) spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de [AWS conformité Programmes AWS](#) de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir [Téléchargement de rapports dans AWS Artifact](#) .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez [AWS la documentation de sécurité](#).

Ce AWS produit ou service suit le [modèle de responsabilité partagée](#) par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la [AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité](#).

Résilience pour ce AWS produit ou service

L'infrastructure AWS mondiale est construite autour Régions AWS de zones de disponibilité.

Régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant.

Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section [Infrastructure AWS mondiale](#).

Ce AWS produit ou service suit le [modèle de responsabilité partagée](#) par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la [AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité](#).

Sécurité de l'infrastructure pour ce AWS produit ou service

Ce AWS produit ou service utilise des services gérés et est donc protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section [Sécurité du AWS cloud](#). Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section [Protection de l'infrastructure](#) dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à ce AWS produit ou service via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser [AWS Security Token Service](#) (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Ce AWS produit ou service suit le [modèle de responsabilité partagée](#) par le biais des services Amazon Web Services (AWS) spécifiques qu'il prend en charge. Pour obtenir des informations sur la sécurité des AWS services, consultez la [AWS page de documentation sur la sécuritéAWS des services et les services concernés par les efforts de AWS conformité par programme de conformité](#).

Appliquer une version minimale de TLS dans le AWS SDK pour .NET

Pour renforcer la sécurité lors de la communication avec AWS les services, vous devez configurer le AWS SDK pour .NET pour utiliser le protocole TLS 1.2 ou version ultérieure.

AWS SDK pour .NET Utilise le moteur d'exécution .NET sous-jacent pour déterminer le protocole de sécurité à utiliser. Par défaut, les versions actuelles de .NET utilisent le dernier protocole configuré pris en charge par le système d'exploitation. Votre application peut remplacer ce comportement du kit SDK, mais il n'est pas recommandé de le faire.

.NET Core

Par défaut, .NET Core utilise le dernier protocole configuré pris en charge par le système d'exploitation. Le AWS SDK pour .NET ne fournit pas de mécanisme permettant de remplacer cela.

Si vous utilisez une version .NET Core antérieure à 2.1, nous vous recommandons vivement de mettre à niveau votre version .NET Core.

Pour plus d'informations spécifiques à chaque système d'exploitation, reportez-vous à ce qui suit.

Windows

Dans les distributions modernes de Windows la prise en charge de TLS 1.2 est [activée par défaut](#). Si vous utilisez Windows 7 SP1 ou Windows Server 2008 R2 SP1, vous devez vous assurer que le support TLS 1.2 est activé dans le registre, comme décrit dans <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12>. Si vous exécutez une distribution antérieure, vous devez mettre à niveau votre système d'exploitation. Pour plus d'informations sur la prise en charge du protocole TLS 1.3 sous Windows, consultez la dernière documentation Microsoft pour connaître les versions minimales requises du client ou du serveur.

macOS

Si vous exécutez .NET Core 2.1 ou version ultérieure, TLS 1.2 est activé par défaut. Le protocole TLS 1.2 est pris en charge par [OS X Mavericks 10.9 ou version](#) ultérieure. .NET Core version 2.1 et versions ultérieures nécessitent des versions plus récentes de macOS, comme décrit à l'adresse <https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos>.

Si vous utilisez .NET Core 1.0, .NET Core [utilise OpenSSL sur macOS](#), une dépendance qui doit être installée séparément. OpenSSL a ajouté le support pour TLS 1.2 dans la version 1.0.1, et le support pour TLS 1.3 dans la version 1.1.1.

Linux

.NET Core sous Linux nécessite OpenSSL, qui est fourni avec de nombreuses distributions Linux. Mais il peut également être installé séparément. OpenSSL a ajouté le support pour TLS 1.2 dans la

version 1.0.1, et le support pour TLS 1.3 dans la version 1.1.1. Si vous utilisez une version moderne de .NET Core (2.1 ou version ultérieure) et que vous avez installé un gestionnaire de paquets, il est probable qu'une version plus moderne d'OpenSSL a été installée pour vous.

Pour être sûr, vous pouvez exécuter **openssl version** dans un terminal et vérifier que la version est postérieure à 1.0.1.

.NET Framework

Si vous exécutez une version moderne de .NET Framework (4.7 ou version ultérieure) et une version moderne de Windows (au moins Windows 8 pour les clients, Windows Server 2012 ou version ultérieure pour les serveurs), TLS 1.2 est activé et utilisé par défaut.

Si vous utilisez un environnement d'exécution .NET Framework qui n'utilise pas les paramètres du système d'exploitation (.NET Framework 3.5 à 4.5.2), il AWS SDK pour .NET tentera d'[ajouter la prise en charge de TLS 1.1 et TLS 1.2](#) aux protocoles pris en charge. Si vous utilisez .NET Framework 3.5, cela ne fonctionnera que si le correctif approprié est installé, comme suit :

- Windows 10 version 1511 et Windows Server 2016 — [KB3156421](#)
- Windows 8.1 et Windows Server 2012 R2 — [KB3154520](#)
- Windows Server 2012 — [KB3154519](#)
- Windows 7 SP1 et Server 2008 R2 SP1 — [KB3154518](#)

Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

Si votre application s'exécute sur un .NET Framework plus récent sous Windows 7 SP1 ou Windows Server 2008 R2 SP1, vous devez vous assurer que le support TLS 1.2 est activé dans le registre, comme décrit dans <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12>. Dans les versions plus récentes de Windows il est [activé par défaut](#).

Pour connaître les meilleures pratiques détaillées relatives à l'utilisation du protocole TLS avec .NET Framework, consultez l'article Microsoft à <https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls> l'adresse.

Outils AWS pour PowerShell

[Outils AWS pour PowerShell](#) utilisez le AWS SDK pour .NET pour tous les appels aux AWS services. Le comportement de votre environnement dépend de la version de Windows que PowerShell vous utilisez, comme suit.

Windows PowerShell 2.0 à 5.x

Windows PowerShell 2.0 à 5.x s'exécute sur .NET Framework. Vous pouvez vérifier quel environnement d'exécution .NET (2.0 ou 4.0) est utilisé à PowerShell l'aide de la commande suivante.

```
$PSVersionTable.CLRVersion
```

- Lorsque vous utilisez .NET Runtime 2.0, suivez les instructions fournies précédemment concernant le AWS SDK pour .NET et .NET Framework 3.5.

Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

- Lorsque vous utilisez .NET Runtime 4.0, suivez les instructions fournies précédemment concernant AWS SDK pour .NET et .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 et les versions ultérieures s'exécutent sur .NET Core. Vous pouvez vérifier quelle version de .NET Core est utilisée en exécutant la commande suivante.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.FrameworkName], $true).FrameworkName
```

Suivez les instructions fournies précédemment concernant AWS SDK pour .NET la version appropriée de .NET Core.

Xamarin

Pour Xamarin, consultez les instructions sur -layer-security. <https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport> Pour résumer :

Pour Android

- Exige Android 5.0 ou version ultérieure.
- Propriétés du projet, options Android : l' HttpClient implémentation doit être définie sur Android et l' SSL/TLS implémentation définie sur Native TLS 1.2+.

Pour iOS

- Exige iOS 7 ou version ultérieure.
- Propriétés du projet, version iOS : l' HttpClientimplémentation doit être définie sur NSURLSession.

Pour macOS

- Exige macOS 10.9 ou version ultérieure.
- Options du projet, build, build Mac : l' HttpClient implémentation doit être définie sur NSURLSession.

Unity

Vous devez utiliser Unity 2018.2 ou version ultérieure et utiliser le runtime de script équivalent .NET 4.x. Vous pouvez le définir dans les paramètres du projet, dans la configuration, dans le lecteur, comme décrit dans le <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgradefichier.html>. L'environnement d'exécution de script équivalent .NET 4.x permet la prise en charge du protocole TLS 1.2 sur toutes les plateformes Unity exécutant Mono ou CPP. IL2

Navigateur (pour Blazor WebAssembly)

WebAssembly s'exécute dans le navigateur plutôt que sur le serveur et utilise le navigateur pour gérer le trafic HTTP. Par conséquent, la prise en charge de TLS est déterminée par la prise en charge du navigateur.

[Blazor WebAssembly, en version préliminaire d'ASP.NET Core 3.1, n'est pris en charge que dans les navigateurs compatibles WebAssembly, comme décrit dans -platforms. https://learn.microsoft.com/en-us/aspnet/core/blazor/supported](https://learn.microsoft.com/en-us/aspnet/core/blazor/supported) Tous les principaux navigateurs étaient compatibles avec le protocole TLS 1.2 avant de le prendre en charge WebAssembly. Si tel est le cas pour votre navigateur, alors si votre application s'exécute, elle peut communiquer via TLS 1.2.

Consultez la documentation de votre navigateur pour plus d'informations et de vérifications.

Migration du client de chiffrement Amazon S3 (V1 vers V2)

Note

Si vous utilisez la version 2 et que vous souhaitez migrer vers la version 4, consultez [Migration du client de chiffrement Amazon S3 \(V2 vers V4\)](#).

Cette rubrique explique comment migrer vos applications de la version 1 (V1) du client de chiffrement Amazon Simple Storage Service (Amazon S3) vers la version 2 (V2), et comment garantir la disponibilité des applications tout au long du processus de migration.

Les objets chiffrés avec le client V2 ne peuvent pas être déchiffrés avec le client V1. Afin de faciliter la migration vers le nouveau client sans avoir à rechiffrer tous les objets en même temps, un client « V1-Transitional » a été fourni. Ce client peut déchiffrer les objets chiffrés en V1 et en V2, mais chiffre les objets uniquement dans un format compatible avec la version 1. Le client V2 peut déchiffrer à la fois les objets chiffrés en V1 et en V2 (lorsqu'il est activé pour les objets V1), mais chiffre les objets uniquement dans un format compatible avec la version v2.

Présentation de la migration

Cette migration s'effectue en trois phases. Ces phases sont présentées ici et décrites en détail ultérieurement. Chaque phase doit être terminée pour tous les clients qui utilisent des objets partagés avant le début de la phase suivante.

1. Mettez à jour les clients existants vers les clients de transition V1 pour lire les nouveaux formats. Tout d'abord, mettez à jour vos applications pour qu'elles dépendent du client V1-Transitional plutôt que du client V1. Le client V1-Transitional permet à votre code existant de déchiffrer les objets écrits par les nouveaux clients V2 et les objets écrits dans un format compatible avec la version 1.

 Note

Le client V1-Transitional est fourni à des fins de migration uniquement. Procédez à la mise à niveau vers le client V2 après avoir migré vers le client V1-Transitional.

2. Migrez les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats. Remplacez ensuite tous les clients de transition V1 de vos applications par des clients V2 et définissez le profil de sécurité sur `V2AndLegacy`. La définition de ce profil de sécurité sur les clients V2 permet à ces clients de déchiffrer des objets chiffrés dans un format compatible avec la version 1.
3. Mettez à jour les clients V2 pour qu'ils ne lisent plus les formats V1. Enfin, une fois que tous les clients ont été migrés vers la version V2 et que tous les objets ont été chiffrés ou rechiffrés dans un format compatible avec la version 2, définissez le profil de sécurité V2 sur au lieu de `V2AndLegacy`. Cela empêche le déchiffrement d'objets au format compatible avec la version 1.

Mettez à jour les clients existants vers des clients de transition vers la version 1 pour lire les nouveaux formats

Le client de chiffrement V2 utilise des algorithmes de chiffrement que les anciennes versions du client ne prennent pas en charge. La première étape de la migration consiste à mettre à jour vos clients de déchiffrement V1 afin qu'ils puissent lire le nouveau format.

Le client V1-Transitional permet à vos applications de déchiffrer les objets chiffrés en V1 et en V2. Ce client fait partie du package [NuGet Amazon.Extensions.S3.Encryption](#). Effectuez les étapes suivantes sur chacune de vos applications pour utiliser le client V1-Transitional.

1. Prenez une nouvelle dépendance vis-à-vis du package [Amazon.Extensions.S3.Encryption](#). Si votre projet dépend directement du `AWSSDK.S3` ou `AWSSDK.KeyManagementService`, vous devez soit mettre à jour ces dépendances, soit les supprimer afin que leurs versions mises à jour soient intégrées à ce nouveau package.
2. Remplacez l'instruction `using` appropriée par « `Amazon.S3.Encryption` » à « `Amazon.Extensions.S3.Encryption` », comme suit :

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. Reconstituez et redéployez votre application.

Le client V1-Transitional est entièrement compatible avec l'API du client V1, aucune autre modification de code n'est donc requise.

Migrer les clients de transition V1 vers les clients V2 pour écrire de nouveaux formats

Le client V2 fait partie du package [NuGet Amazon.Extensions.S3.Encryption](#). Il permet à vos applications de déchiffrer les objets chiffrés en V1 et en V2 (si cela est configuré pour le faire), mais chiffre les objets uniquement dans un format compatible avec la version v2.

Après avoir mis à jour vos clients existants pour lire le nouveau format de chiffrement, vous pouvez procéder à la mise à jour en toute sécurité de vos applications vers les clients de chiffrement et de déchiffrement V2. Pour utiliser le client V2, effectuez les étapes suivantes sur chacune de vos applications :

1. Remplacez `EncryptionMaterials` par `EncryptionMaterialsV2`
 - a. Lorsque vous utilisez KMS :
 - i. Fournissez un ID de clé KMS.
 - ii. Déclarez la méthode de cryptage que vous utilisez, c'est-à-dire `KmsType.KmsContext`.
 - iii. Fournissez un contexte de chiffrement à KMS à associer à cette clé de données. Vous pouvez envoyer un dictionnaire vide (le contexte de chiffrement Amazon sera toujours intégré), mais il est recommandé de fournir un contexte supplémentaire.
 - b. Lorsque vous utilisez des méthodes d'encapsulation de clé fournies par l'utilisateur (chiffrement symétrique ou asymétrique) :
 - i. Fournissez une AES ou une RSA instance contenant le matériel de chiffrement.
 - ii. Déclarez l'algorithme de chiffrement à utiliser, c'est-à-dire `SymmetricAlgorithmType.AesGcm` ou `AsymmetricAlgorithmType.RsaOaepSha1`.
2. Passez `AmazonS3CryptoConfiguration` à `AmazonS3CryptoConfigurationV2` avec la `SecurityProfile` propriété définie sur `SecurityProfile.V2AndLegacy`.

3. Remplacez `AmazonS3EncryptionClient` par `AmazonS3EncryptionClientV2`. Ce client prend les objets récemment convertis `AmazonS3CryptoConfigurationV2` et `EncryptionMaterialsV2` les objets des étapes précédentes.

Exemple : KMS vers KMS+Context

Prémigration

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var encryptionMaterial = new
    EncryptionMaterials("1234abcd-12ab-34cd-56ef-1234567890ab");
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new
    EncryptionMaterialsV2("1234abcd-12ab-34cd-56ef-1234567890ab", KmsType.KmsContext,
        encryptionContext);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
    encryptionMaterial);
```

Exemple : algorithme symétrique (enroulement de clé AES-CBC vers AES-GCM)

`StorageMode` peut avoir la valeur `ObjectMetadata` ou `InstructionFile`.

Prémigration

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterials(symmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
    SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
    encryptionMaterial);
```

Note

Lors du déchiffrement avec AES-GCM, lisez l'objet dans son intégralité avant de commencer à utiliser les données déchiffrées. Cela permet de vérifier que l'objet n'a pas été modifié depuis qu'il a été chiffré.

Exemple : algorithme asymétrique (RSA to RSA-OAEP-SHA 1 Key Wrap)

StorageMode peut avoir la valeur ObjectMetadata ou InstructionFile.

Prémigration

```
using System.Security.Cryptography;
```



```
using Amazon.S3.Encryption;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterials(asymmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Après la migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterialsV2(asymmetricAlgorithm,
    AsymmetricAlgorithmType.RsaOaepSha1);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
    encryptionMaterial);
```

Mettre à jour les clients V2 pour ne plus lire les formats V1

À terme, tous les objets auront été chiffrés ou rechiffrés à l'aide d'un client V2. Une fois cette conversion terminée, vous pouvez désactiver la compatibilité V1 dans les clients V2 en définissant la `SecurityProfile` propriété sur `SecurityProfile.V2`, comme indiqué dans l'extrait suivant.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migration du client de chiffrement Amazon S3 (V2 vers V4)

Note

Si vous utilisez la V1 et que vous souhaitez migrer vers la V4, vous devez d'abord migrer vers la V2. Consultez [Migration du client de chiffrement Amazon S3 \(V1 vers V2\)](#).

Cette rubrique explique comment migrer vos applications de la version 2 (V2) du client de chiffrement Amazon Simple Storage Service (Amazon S3) vers la version 4 (V4) et comment garantir la disponibilité des applications tout au long du processus de migration. La V4 utilise l'AES-GCM avec un engagement clé pour le chiffrement du contenu et introduit des politiques d'engagement pour améliorer la sécurité contre les attaques par substitution de clés.

Le client V4 est disponible dans le package [NuGet Amazon.Extensions.S3.Encryption](#).

Important

Changements majeurs : les clients V4 configurés avec une `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` politique ne peuvent pas déchiffrer les objets chiffrés avec les clients V1 ou V2. Seuls les derniers clients V2 peuvent déchiffrer les objets chiffrés V4 avec un engagement clé. Avant de passer à cette `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` politique, vous devez rechiffrer toutes vos données existantes à l'aide de clients V4 avec activation de la clé d'engagement.

Comprendre les concepts de la V4

La V4 utilise l'AES-GCM avec un engagement clé en matière de chiffrement de contenu et introduit des concepts de sécurité clés qui améliorent la protection de vos données chiffrées :

Politique d'engagement

La politique d'engagement contrôle la manière dont le client de chiffrement gère l'engagement des clés lors des opérations de chiffrement et de déchiffrement. V4 soutient trois politiques d'engagement :

FORBID_ENCRYPT_ALLOW_DECRYPT

Chiffrement : sans engagement

Déchiffrement : autorise les objets non validants

Sécurité : n'impose pas d'engagement et peut permettre la falsification

Compatibilité : toutes les implémentations V2 et V4 peuvent lire les objets chiffrés avec cette politique

REQUIRE_ENCRYPT_ALLOW_DECRYPT

Chiffrement : avec engagement clé

Déchiffrement : autorise à la fois les objets validants et non validants

Sécurité : les nouveaux objets sont protégés contre les attaques de substitution de clés, les anciens objets restent lisibles

Compatibilité : seule la version V4 prend en charge cette politique

REQUIRE_ENCRYPT_REQUIRE_DECRYPT (Par défaut pour la V4)

Chiffrement : avec engagement clé

Déchiffrement : validation d'objets uniquement

Sécurité : mise en œuvre complète des engagements pour une sécurité maximale

Compatibilité : seule la version V4 prend en charge cette politique

AES GCM avec un engagement clé

La V4 utilise l'AES-GCM avec un engagement clé pour le chiffrement du contenu, ce qui améliore la sécurité :

- Protection contre la falsification : elle protège contre les attaques de substitution de clés en liant cryptographiquement la clé aux données cryptées.
- Compatibilité des versions : les objets chiffrés avec un engagement clé ne peuvent être déchiffrés que par les clients V4 et les versions plus récentes.

⚠ Warning

Avant d'activer le chiffrement par engagement par clé en production, assurez-vous que toutes les applications qui ont besoin de déchiffrer vos objets ont été mises à niveau vers la version V4 ou une version ultérieure, car le client V2 est obsolète.

Mettre à jour les clients existants pour lire les formats V4

Le client de chiffrement V4 utilise des algorithmes de chiffrement que les anciennes versions du client ne prennent pas en charge. La première étape de la migration consiste à mettre à jour vos clients V2 afin qu'ils puissent lire le nouveau format V4.

Mettre à jour les dépendances NuGet du package

Mettez à jour vos applications pour utiliser la dernière version du package

[Amazon.Extensions.S3.Encryption](#) NuGet qui inclut le support V4. Effectuez les étapes suivantes sur chacune de vos applications :

1. Effectuez la mise à jour vers le dernier package [Amazon.Extensions.S3.Encryption](#). Si votre projet dépend directement du AWSSDK.S3 ou AWSSDK.KeyManagementServicepackages, vous devez soit mettre à jour ces dépendances, soit les supprimer afin que leurs versions mises à jour soient intégrées à ce nouveau package.
2. Assurez-vous que vos `using` instructions font référence au bon espace de noms :

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;
```

3. Reconstituez et redéployez votre application.

Vos clients V2 existants continueront de fonctionner avec le package mis à jour et pourront déchiffrer les objets chiffrés par les clients V4 (en fonction de la politique d'engagement utilisée).

Création et déploiement d'applications

Après avoir mis à jour les dépendances de votre NuGet package :

1. Créez votre application pour vous assurer que toutes les dépendances sont correctement résolues.

2. Testez votre application dans un environnement de développement pour vérifier que les fonctionnalités existantes continuent de fonctionner.
3. Déployez l'application mise à jour dans votre environnement de production.

Cette mise à jour permet à vos clients V2 existants de déchiffrer les objets qui seront chiffrés par les clients V4, garantissant ainsi la compatibilité pendant le processus de migration.

Migrer vers des clients V4

Après avoir mis à jour vos clients existants pour lire le nouveau format de chiffrement, vous pouvez procéder à la mise à jour sécurisée de vos applications afin d'utiliser les clients de chiffrement et de déchiffrement V4. Le client V4 fournit une sécurité renforcée grâce à un engagement clé tout en maintenant la compatibilité avec les objets chiffrés existants.

Processus de migration en 4 étapes

La migration de la V2 vers la V4 suit un processus structuré en 4 étapes pour garantir la compatibilité et la sécurité. Chaque étape représente une configuration spécifique qui doit être déployée dans toutes vos applications avant de passer à l'étape suivante.

1. Étape 0 : Client V2 (point de départ) - Votre implémentation V2 existante
2. Étape 1 : compatibilité V4 avec V2 - Migrer vers le client V4 tout en conservant un comportement de chiffrement compatible avec la version V2
3. Étape 2 : version V4 avec écriture par clé - Commencez à chiffrer avec un engagement par clé tout en autorisant le déchiffrement des objets existants
4. Étape 3 : V4 avec mise en œuvre complète - Nécessite un engagement clé pour le chiffrement et le déchiffrement

Étape 0 : Client V2 (point de départ)

Cela représente la configuration de votre client V2 existante. Cette étape montre l'état de départ avant la migration.

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;  
using Amazon.S3.Model;
```

```
// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
    encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
    encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Étape 1 : compatibilité V4 avec V2

Migrez vers le client V4 tout en conservant un comportement identique à celui de la V2. Cette étape utilise une `FORBID_ENCRYPT_ALLOW_DECRYPT` politique pour chiffrer sans engagement et permettre le déchiffrement de tous les objets.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
    encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
    encryptionMaterial);
```

```
// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportement : chiffre sans engagement, peut déchiffrer à la fois les objets validants et non validants. Comportement identique à celui de la V2.

Étape 2 : V4 avec des écritures d'engagement clés

Commencez à chiffrer avec un engagement clé tout en maintenant la rétrocompatibilité pour le déchiffrement. Cette étape utilise `REQUIRE_ENCRYPT_ALLOW_DECRYPT` la politique.

Warning

Avant de déployer l'étape 2, assurez-vous que tous les lecteurs ont été mis à jour à l'étape 1 ou à une version ultérieure pour gérer le chiffrement par engagement par clé.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
    encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
    encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
```

```
    BucketName = bucket,  
    Key = objectKey,  
    ContentBody = content  
});
```

Comportement : chiffre avec engagement, permet de déchiffrer à la fois les objets validants et non validants. Les nouveaux objets sont protégés contre les attaques de substitution de clés.

Étape 3 : V4 avec mise en œuvre complète

Exigez un engagement clé pour le chiffrement et le déchiffrement. Cette étape utilise une `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` politique pour une sécurité maximale.

Warning

Avant de déployer l'étape 3, assurez-vous que tous les objets de votre système ont été rechiffrés avec une clé d'engagement (étape 2). Cette étape ne parviendra pas à déchiffrer les objets chiffrés sans engagement.

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;  
using Amazon.S3.Model;  
  
// Step 3: V4 Client with full key commitment enforcement  
var encryptionContext = new Dictionary<string, string>();  
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,  
    encryptionContext);  
  
var configuration = new AmazonS3CryptoConfigurationV4(  
    SecurityProfile.V4,  
    CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT,  
    ContentEncryptionAlgorithm.AesGcmWithCommitment);  
  
var encryptionClient = new AmazonS3EncryptionClientV4(configuration,  
    encryptionMaterial);  
  
// Use the client for PutObject and GetObject operations  
await encryptionClient.PutObjectAsync(new PutObjectRequest  
{  
    BucketName = bucket,
```



```
    Key = objectKey,  
    ContentBody = content  
});
```

Comportement : chiffre avec engagement, déchiffre uniquement les objets chiffrés avec engagement.
Sécurité maximale contre les attaques par substitution de clés.

Exemples de configuration supplémentaires

Cette section fournit des exemples supplémentaires pour configurer les clients V4 avec différentes options lors de la migration.

Permettre le support existant

Pour permettre aux clients V4 de lire les objets chiffrés par les clients V1 et V2, configurez le client avec une politique d'engagement qui autorise le déchiffrement traditionnel :

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;  
  
// Configure V4 client to read V1/V2 objects  
var configuration = new  
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)  
{  
    StorageMode = CryptoStorageMode.ObjectMetadata  
};  
  
// This configuration allows:  
// - Encryption: With commitment (secure)  
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Utilisez cette configuration lors de la migration lorsque vous devez déchiffrer des objets chiffrés par des clients plus anciens tout en vous assurant que les nouveaux objets sont chiffrés avec une sécurité renforcée.

Configuration de la méthode de stockage

La V4 prend en charge deux méthodes de stockage pour les métadonnées de chiffrement.
Choisissez la méthode la mieux adaptée à votre cas d'utilisation :

Métadonnées d'objet (par défaut)

```
var configuration = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Fichier d'instructions

```
var configuration = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

À utiliser `InstructionFile` lorsque vous devez conserver les métadonnées d'objets à d'autres fins ou lorsque vous travaillez avec des objets dont la taille des métadonnées est limitée.

Configuration de la politique d'engagement

Choisissez la politique d'engagement appropriée en fonction de vos exigences de sécurité et de la phase de migration :

Phase de migration (compatibilité V2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT);
```

Phase de transition (recommandée)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT);
```

Phase de sécurité complète

```
// For maximum security: require commitment for both encryption and decryption
```

```
var secureConfig = new  
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT);
```

Commencez par FORBID_ENCRYPT_ALLOW_DECRYPT la migration initiale, passez à la phase REQUIRE_ENCRYPT_ALLOW_DECRYPT de transition, puis utilisez-la REQUIRE_ENCRYPT_REQUIRE_DECRYPT lorsque tous les clients ont été mis à niveau et que tous les objets ont été rechiffrés avec engagement.

Considérations spéciales relatives au AWS SDK pour .NET

Cette section contient des considérations relatives aux cas particuliers dans lesquels les configurations ou procédures normales ne sont pas appropriées ou suffisantes.

Rubriques

- [Obtention d'assemblages pour le AWS SDK pour .NET](#)
- [Accès aux informations d'identification et aux profils dans une application](#)
- [Considérations spéciales relatives au support Unity](#)
- [Considérations spéciales pour le support Xamarin](#)

Obtention d'assemblages pour le AWS SDK pour .NET

Cette rubrique décrit comment vous pouvez obtenir les AWSSDK assemblages et les stocker localement (ou sur site) pour les utiliser dans vos projets. Cette méthode n'est pas recommandée pour gérer les références au SDK, mais elle est requise dans certains environnements.

Note

La méthode recommandée pour gérer les références aux SDK consiste à télécharger et à installer uniquement les NuGet packages dont chaque projet a besoin. Cette méthode est décrite dans [Installez AWSSDK des packages avec NuGet](#).

Si vous ne pouvez pas ou n'êtes pas autorisé à télécharger et à installer des NuGet packages par projet, les options suivantes sont à votre disposition.

Téléchargez et extrayez des fichiers ZIP

(N'oubliez pas que ce n'est pas la [méthode recommandée](#) pour gérer les références à AWS SDK pour .NET.)

1. Téléchargez l'un des fichiers ZIP suivants :

- [aws-sdk-net8.0.zip](#) - Assemblages compatibles avec .NET 8 et versions ultérieures.
- [aws-sdk-netcoreapp3.1.zip](#) - Assemblages compatibles avec .NET Core 3.1 et versions ultérieures.

- [aws-sdk-netstandard2.0.zip](#) - Assemblages compatibles avec .NET Standard 2.0 et 2.1.
- [aws-sdk-net45.zip](#) : assemblages compatibles avec .NET Framework 4.5 et versions ultérieures.
- [aws-sdk-net35.zip](#) : assemblages compatibles avec .NET Framework 3.5.

 Warning

À compter du 15 août 2024, le AWS SDK pour .NET support de .NET Framework 3.5 cessera et la version minimale de .NET Framework sera remplacée par 4.7.2. Pour plus d'informations, consultez le billet de blog [Changements importants à venir pour les cibles .NET Framework 3.5 et 4.5 du AWS SDK pour .NET](#).

2. Extrayez les assemblages dans un dossier de « téléchargement » de votre système de fichiers, peu importe où. Prenez note de ce dossier.
3. Lorsque vous configurez votre projet, vous obtenez les assemblages requis dans ce dossier, comme décrit dans [Installez AWSSDK des assemblages sans NuGet](#).

Accès aux informations d'identification et aux profils dans une application

La méthode préférée pour utiliser les informations d'identification est de les autoriser AWS SDK pour .NET à les rechercher et à les récupérer pour vous, comme décrit dans [Résolution des informations d'identification et des profils](#).

Toutefois, vous pouvez également configurer votre application pour récupérer activement les profils et les informations d'identification, puis utiliser explicitement ces informations d'identification lors de la création d'un client de AWS service.

Pour récupérer activement les profils et les informations d'identification, utilisez les classes d'[Amazon.Runtime.CredentialManagement](#) espace de noms.

- Pour rechercher un profil dans un fichier utilisant le format de fichier AWS d'informations d'identification (soit le [fichier AWS d'informations d'identification partagé dans son emplacement par défaut](#), soit un fichier d'informations d'identification personnalisé), utilisez la [SharedCredentialsFile](#) classe. Les fichiers de ce format sont parfois simplement appelés fichiers d'identification dans ce texte par souci de concision.

- Pour rechercher un profil dans le SDK Store, utilisez la classe [Net SDKCredentials File](#).
- Pour effectuer une recherche à la fois dans un fichier d'informations d'identification et dans le SDK Store, en fonction de la configuration d'une propriété de classe, utilisez la [CredentialProfileStoreChain](#) classe.

Vous pouvez utiliser cette classe pour rechercher des profils. Vous pouvez également utiliser cette classe pour demander des AWS informations d'identification directement au lieu d'utiliser la `AWSCredentialsFactory` classe (décrite ci-dessous).

- Pour récupérer ou créer différents types d'informations d'identification à partir d'un profil, utilisez la classe [AWSCredentialsFactory](#).

Les sections suivantes fournissent des exemples pour ces classes.

Exemples de cours CredentialProfileStoreChain

Vous pouvez obtenir des informations d'identification ou des profils auprès de la [CredentialProfileStoreChain](#) classe à l'aide [TryGetProfile](#) des méthodes [TryGetAWSCredentials](#) or. La `ProfilesLocation` propriété de la classe détermine le comportement des méthodes, comme suit :

- Si la valeur `ProfilesLocation` est nulle ou vide, recherchez le SDK Store si la plateforme le prend en charge, puis recherchez le fichier AWS d'informations d'identification partagé à l'emplacement par défaut.
- Si la `ProfilesLocation` propriété contient une valeur, recherchez le fichier d'informations d'identification spécifié dans la propriété.

Obtenez des informations d'identification depuis le SDK Store ou le fichier AWS d'informations d'identification partagé

Cet exemple vous montre comment obtenir des informations d'identification à l'aide de la `CredentialProfileStoreChain` classe, puis les utiliser pour créer un objet [AmazonS3Client](#). Les informations d'identification peuvent provenir du SDK Store ou du fichier d' AWS informations d'identification partagé à l'emplacement par défaut.

Cet exemple utilise également le logiciel [Amazon.Runtime.AWSCredentials](#) classe.

```
var chain = new CredentialProfileStoreChain();
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("some_profile", out awsCredentials))
```

```
{
    // Use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Obtenir un profil depuis le SDK Store ou le fichier d' AWS informations d'identification partagé

Cet exemple montre comment obtenir un profil à l'aide de la `CredentialProfileStoreChain` classe. Les informations d'identification peuvent provenir du SDK Store ou du fichier d' AWS informations d'identification partagé à l'emplacement par défaut.

Cet exemple utilise également la [CredentialProfile](#) classe.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Obtenir des informations d'identification à partir d'un fichier d'informations d'identification personnalisé

Cet exemple montre comment obtenir des informations d'identification à l'aide de la `CredentialProfileStoreChain` classe. Les informations d'identification proviennent d'un fichier qui utilise le format de fichier AWS d'informations d'identification mais qui se trouve à un autre emplacement.

Cet exemple utilise également le logiciel [Amazon.Runtime.AWSCredentials](#) classe.

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials to create an AWS service client
}
```

Exemples pour les classes SharedCredentialsFile et AWSCredentials Factory

Créez un client Amazon S3 à l'aide de la classe SharedCredentialsFile

Cet exemple vous montre comment rechercher un profil dans le fichier d'AWS informations d'identification partagé, créer des informations AWS d'identification à partir du profil, puis utiliser les informations d'identification pour créer un objet [AmazonS3Client](#). L'exemple utilise la [SharedCredentialsFile](#) classe.

Cet exemple utilise également la [CredentialProfile](#) classe et le [fichier Amazon.Runtime.AWSCredentials](#) classe.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
    awsCredentials))
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Note

La classe [Net SDKCredentials File](#) peut être utilisée exactement de la même manière, sauf que vous instanciez un nouvel objet Net SDKCredentials File au lieu d'un SharedCredentialsFile objet.

Considérations spéciales relatives au support Unity

Lorsque vous utilisez [.NET Standard 2.0](#) pour votre application Unity, celle-ci doit référencer directement les AWS SDK pour .NET assemblés (fichiers DLL) plutôt que d'utiliser NuGet. AWS SDK pour .NET Compte tenu de cette exigence, vous devrez effectuer les actions importantes suivantes.

- Vous devez obtenir les AWS SDK pour .NET assemblages et les appliquer à votre projet. Pour plus d'informations sur la procédure à suivre, reportez-vous [Téléchargez et extrayez des fichiers ZIP](#) à la rubrique [Obtention d' AWSSDK assemblages](#).
- Vous devez inclure les éléments suivants DLLs dans votre projet Unity en plus de DLLs for AWSSDK.Core et des autres AWS services que vous utilisez. À partir de la version 3.5.109 du AWS SDK pour .NET, le fichier ZIP standard .NET contient ces informations supplémentaires.
DLLs
 - [Microsoft Bcl. AsyncInterfaces.dll](#)
 - [Système. Runtime. CompilerServices.Unsafe.dll](#)
 - [System.Threading.Tasks.Extensions.dll](#)
- Si vous utilisez [IL2CPP](#) pour créer votre projet Unity, vous devez ajouter un `link.xml` fichier dans votre dossier Asset pour empêcher le découpage du code. Le `link.xml` fichier doit répertorier tous les AWSSDK assemblages que vous utilisez, et chacun doit inclure l'`preserve="all"` attribut. L'extrait suivant montre un exemple de ce fichier.

```
<linker>
  <assembly fullname="AWSSDK.Core" preserve="all"/>
  <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
  <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

Pour lire des informations générales intéressantes relatives à cette exigence, consultez l'article sur <https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or>.

Outre ces considérations spéciales, consultez [Ce qui a changé pour la version 3.5](#) pour plus d'informations sur la migration de votre application Unity vers la version 3.5 du AWS SDK pour .NET.

Considérations spéciales pour le support Xamarin

Les projets Xamarin (nouveaux et existants) doivent cibler .NET Standard 2.0. Consulter [Prise en charge de .NET Standard 2.0 dans Xamarin.Forms](#) et [Prise en charge de l'implémentation .NET](#).

Consultez également les informations sur [Portable Class Library et Xamarin](#).

Référence d'API pour AWS SDK pour .NET

AWS SDK pour .NET fournit une API que vous pouvez utiliser pour accéder aux AWS services. Pour connaître les classes et les méthodes disponibles dans l'API, consultez la [référence de l'AWS SDK pour .NET API](#).

Outre la référence générale donnée ci-dessus, chacun des exemples de [Exemples de code avec conseils](#) cette section contient des références aux méthodes et classes spécifiques utilisées dans cet exemple.

À propos des versions de référence de l'API

La référence d'API décrite précédemment concerne les versions 3.0 et ultérieures du AWS SDK pour .NET.

Pour plus d'informations sur la migration à partir d'anciennes versions du SDK, voir [Migrez votre projet](#)

Pour trouver le contenu obsolète des versions antérieures de la référence d'API du SDK, consultez les éléments suivants :

- [AWS SDK pour .NET API Reference V1 \(obsolète\)](#)
- [AWS SDK pour .NET API Reference V2 \(obsolète\)](#)

Pour afficher une référence d' AWS SDK pour .NET API obsolète, vous devez l'extraire et configurer un navigateur Web. Les instructions ci-dessous sont des exemples illustrant la procédure à suivre. Ils sont basés sur l'utilisation du navigateur Web Google Chrome sur un système Windows. Adaptez-les à votre navigateur Web et à votre système d'exploitation spécifiques.

Afficher la référence d'API obsolète V1

1. Téléchargez le fichier ZIP de la version obsolète de l' AWS SDK pour .NET API Reference V1. Par défaut, le nom du fichier ZIP est `sdksfor-net-api-ref_v1_deprecated.zip`. Ce nom sera utilisé tout au long de ces instructions.
2. Placez le fichier ZIP dans le dossier de votre choix. Pour ces instructions, le nom du dossier est supposé être `C:\work\temp\api-refs\V1`.

3. Cliquez avec le bouton droit sur le fichier ZIP et choisissez Extraire tout. Acceptez l'emplacement par défaut, qui correspond C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1 à ces instructions.
4. Créez un raccourci pour Google Chrome dans le C:\work\temp\api-refs\V1 dossier. Veillez à ne pas déplacer l'application d'origine.
5. Dans les propriétés du nouveau raccourci, définissez les champs suivants :
 - Cible : "*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"

Pour l'--user-data-dirargument, utilisez un nom de dossier adapté à votre environnement. Il n'est pas nécessaire que le dossier existe.

 - Commencez dans : C:\work\temp\api-refs\V1
6. Donnez un nom approprié au raccourci.
7. Ouvrez le raccourci pour afficher l'ancienne référence d'API.

Afficher la référence d'API obsolète V2

1. Téléchargez le fichier ZIP de la version obsolète de l' AWS SDK pour .NET API Reference V2. Par défaut, le nom du fichier ZIP estsdkfornet-api-ref_v2_deprecated.zip. Ce nom sera utilisé tout au long de ces instructions.
2. Placez le fichier ZIP dans le dossier de votre choix. Pour ces instructions, le nom du dossier est supposé êtreC:\work\temp\api-refs\V2.
3. Cliquez avec le bouton droit sur le fichier ZIP et choisissez Extraire tout. Acceptez l'emplacement par défaut, qui correspond C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2 à ces instructions.
4. Créez un raccourci pour Google Chrome dans le C:\work\temp\api-refs\V2 dossier. Veillez à ne pas déplacer l'application d'origine.
5. Dans les propriétés du nouveau raccourci, définissez les champs suivants :
 - Cible : "*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"

Pour l'- -user-data-dirargument, utilisez un nom de dossier adapté à votre environnement. Il n'est pas nécessaire que le dossier existe.

- Commencez dans : C:\work\temp\api-refs\V2
6. Donnez un nom approprié au raccourci.
 7. Ouvrez le raccourci pour afficher l'ancienne référence d'API.

Historique du document

Le tableau suivant décrit les modifications importantes apportées depuis la dernière version du Guide du AWS SDK pour .NET développeur. Pour recevoir les notifications concernant les mises à jour de cette documentation, abonnez-vous à un [flux RSS](#).

Modification	Description	Date
Quoi de neuf	Ajout d'une note concernant la version 2.0 de l'outil de AWS déploiement pour .NET CLI.	17 octobre 2025
AWS OpsWorks	AWS OpsWorks a atteint la fin de sa vie. Il a été retiré de la documentation.	16 septembre 2025
Quoi de neuf	End-of-support a été annoncé pour la V3 du AWS SDK pour .NET.	21 août 2025
Cadre de traitement des messages	Le cadre de traitement des AWS messages pour .NET est désormais disponible pour tous !	5 mai 2025
Quoi de neuf	La version 4 du AWS SDK pour .NET est généralement disponible ! Pour plus d'informations, consultez le guide du développeur AWS SDK pour .NET (V4) , en particulier la rubrique relative à la migration vers la version 4 .	28 avril 2025
Rétentatives et délais d'attente	Informations incluses sur les délais d'expiration par méthode utilisés Cancellat	9 avril 2025

ionToken pour les appels de méthode asynchrones.

[Intégration AWS à .NET Aspire dans AWS SDK pour .NET](#)

Vous trouverez des informations sur la façon dont l'intégration à .NET Aspire AWS a été intégrée dans le AWS SDK pour .NET.

15 février 2025

[Observabilité](#)

Annnonce de la sortie GA pour l'observabilité

10 février 2025

[Quoi de neuf](#)

Ajout d'informations sur le nouveau comportement par défaut pour la protection de l'intégrité.

15 janvier 2025

[Migration du client de chiffrement S3 V4](#)

Ajout d'informations sur la migration de la V2 vers la V4.

9 décembre 2024

[Quoi de neuf](#)

Ajout d'informations sur la quatrième version préliminaire de la AWS SDK pour .NET version 4.

15 novembre 2024

[Observabilité](#)

Ajout d'informations d'aperçu sur l'observabilité dans le AWS SDK pour .NET, qui permettent de collecter des données de télémétrie.

13 septembre 2024

Référence d'API pour le AWS SDK pour .NET	Les références AWS SDK pour .NET d'API pour les versions V1 et V2 sont devenues obsolètes. Des informations ont été incluses sur cette dépréciation et sur la manière d'obtenir et de visualiser les références obsolètes.	4 septembre 2024
Quoi de neuf	Ajout d'informations sur la première version préliminaire de la AWS SDK pour .NET version 4.	16 août 2024
Quoi de neuf	Informations mises à jour sur les modifications à venir concernant le support du .NET Framework.	20 juin 2024
Quoi de neuf	Ajout d'informations sur la version préliminaire du cadre de traitement des AWS messages pour .NET	28 mars 2024
Quoi de neuf	Informations incluses sur le support pour .NET 8.	23 février 2024
Quoi de neuf	Informations incluses sur les modifications à venir apportées à la prise en charge du .NET Framework.	18 février 2024
Obtention d' AWSSDK assemblages	Informations incluses sur les assemblages compatibles avec .NET 8 et versions ultérieures.	8 janvier 2024

AWS Framework de traitement des messages pour .NET	Informations incluses sur la version bêta du cadre de traitement des messages.	10 décembre 2023
Utilisation des bases de données NoSQL Amazon DynamoDB	Informations mises à jour sur les modèles de programmation de persistance des documents et des objets. Il est désormais possible d'éviter certaines conditions de latence ou de blocage dues aux comportements de démarrage à froid et de pool de threads.	15 novembre 2023
Inclusion de mises à jour supplémentaires des bonnes pratiques IAM	Mise à jour du guide s'aligner sur les bonnes pratiques IAM. Pour plus d'informations, consultez Bonnes pratiques de sécurité dans IAM .	5 octobre 2023
Obtention d' AWSSDK assemblages	Suppression des informations concernant l'installation du AWS SDK pour .NET à l'aide du programme d' Outils AWS pour Windows installation (c'est-à-dire le MSI), qui est devenu obsolète.	25 septembre 2023
Mises à jour des bonnes pratiques IAM	Mise à jour du guide s'aligner sur les bonnes pratiques IAM. Pour plus d'informations, consultez Bonnes pratiques de sécurité dans IAM .	18 juillet 2023

Annotations Lambda	Le framework d' AWS Lambda annotations a été publié pour une mise à disposition générale.	17 juillet 2023
Quoi de neuf	Ajout d'informations sur la version préliminaire du fournisseur de cache distribué pour DynamoDB.	15 juillet 2023
Table des matières	Table des matières mise à jour pour rendre les exemples de code plus faciles à découvrir.	8 juin 2023
Résolution régionale	Ajout d'informations sur la manière dont le SDK résout une spécification de région manquante.	14 mars 2023
Support pour le MSI	Ajout d'une note concernant la fin du support pour le Outils AWS pour Windows programme d'installation.	06 mars 2023
Annotations Lambda (aperçu)	Aperçu du framework d' AWS Lambda annotations.	22 septembre 2022
Déployez des applications pour AWS	Le contenu principal a été déplacé vers un site GitHub Pages : https://aws.github.io/aws-dotnet-deploy/	28 juin 2022
Retraite - Classique EC2	Ajout de notes concernant le retrait de EC2 -Classic.	13 avril 2022

Authentification unique avec le AWS SDK pour .NET	Ajout d'informations sur l'authentification unique (SSO) lors de l'utilisation du AWS SDK pour .NET	17 mars 2022
Appliquer une version minimale de TLS	Ajout d'informations sur TLS 1.3.	16 mars 2022
Travaillez avec les AWS services	Listes incluses des exemples de code disponibles sur GitHub.	28 février 2022
Activation des métriques du SDK	Suppression des informations relatives à l'activation des métriques du SDK, qui sont devenues obsolètes.	20 janvier 2022
Déployez des applications pour AWS	Ajout d'une référence au AWS Toolkit for Visual Studio, qui fournit des fonctionnalités de déploiement similaires à celles de l'outil de AWS déploiement.	26 octobre 2021
AWS SDK pour .NET consolidation du guide de la version 3	Les deux guides du développeur de la AWS SDK pour .NET version 3, « V3 » et « latest », ont été regroupés en un seul guide sous l'URL « v3 ».	18 août 2021
Migration depuis .NET Standard 1.3	Support pour .NET Standard 1.3 AWS SDK pour .NET est arrivé à sa fin de vie.	25 mars 2021
Déployer des applications vers AWS (version préliminaire)	Ajout d'informations d'aperçu sur l'outil de AWS déploiement, que vous pouvez utiliser pour déployer une application à partir de la CLI .NET.	15 mars 2021

Version 3.5 du AWS SDK pour .NET	La version 3.5 du AWS SDK pour .NET a été publiée.	25 août 2020
Paginateurs	Ajout de paginateurs à de nombreux clients de service, ce qui facilite la pagination des résultats de l'API.	24 août 2020
Rétentatives et délais d'attente	Ajout d'informations sur les modes de nouvelle tentative.	20 août 2020
Utilisation de clés KMS pour le chiffrement S3	Exemple mis à jour pour utiliser la version 2 du client de chiffrement S3.	6 août 2020
Migration depuis .NET Standard 1.3	Ajout d'informations sur l'arrêt de la prise en charge de .NET Standard 1.3 à la fin de l'année 2020.	18 mai 2020
Démarrage rapide	Ajout d'une section de démarrage rapide avec configuration de base et didacticiels pour présenter le kit AWS SDK pour .NET au lecteur.	27 mars 2020
Appliquer le protocole TLS 1.2	Ajout d'informations sur la façon d'appliquer TLS 1.2 dans le SDK.	10 mars 2020