



Guia do desenvolvedor

AWS SDK para .NET (V3)



AWS SDK para .NET (V3): Guia do desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

.....	xiii
O que é o AWS SDK para .NET	1
Sobre essa versão	1
Manutenção e suporte para as versões principais do SDK	2
Casos de uso comuns	2
Tópicos adicionais nesta seção	2
Ferramentas da AWS relacionadas	3
Ferramentas para Windows PowerShell e ferramentas para PowerShell Core	3
Toolkit for VS Code	3
Toolkit para Visual Studio	3
Toolkit for Azure DevOps	4
Plataformas compatíveis	4
.NET Core	4
.NET Standard 2.0	4
.NET Framework 4.5	5
.NET Framework 3.5	5
Biblioteca de classes portátil e Xamarin	6
Suporte ao Unity	6
Mais informações	6
Referência de SDKs e ferramentas	6
Histórico de revisões	6
O que há de novo	7
Recursos adicionais	9
Conceitos básicos	12
Instale e configure seu conjunto de ferramentas	12
Desenvolvimento entre plataformas	13
Windows com Visual Studio e .NET Core	13
Próxima etapa	14
Configure a autenticação do SDK	14
Faça login usando as credenciais do console	14
Habilitar e configurar o Centro de Identidade do IAM	15
Configure o SDK para usar o Centro de Identidade do IAM.	16
Iniciar uma sessão do portal de AWS acesso	17
Mais informações	18

Faça um tour rápido	18
Aplicativo simples para várias plataformas	19
Aplicativo simples baseado no Windows	24
Próximas etapas	30
Iniciar um novo projeto	30
Configurar a região da AWS	32
Crie um cliente de serviço com uma região específica	32
Especifique uma região para todos os clientes de serviço	33
Resolução da região	34
Informações especiais sobre a região da China (Pequim)	35
Informações especiais sobre novos serviços da AWS	35
Instale pacotes do AWSSDK com o NuGet	35
Como usar o NuGet pela linha de comando ou pelo terminal	36
Como usar o NuGet no Visual Studio Solution Explorer	36
Usar o NuGet a partir do Console de gerenciamento de pacotes	37
Instale assemblies do AWSSDK sem o NuGet	38
Resolução de perfil e credenciais	39
Resolução do perfil	40
Usar credenciais da conta de usuário federado	41
Especificar funções ou credenciais temporárias	42
Usar credenciais de proxy	42
Usuários e perfis	43
Usuários e conjuntos de permissões	43
Perfis de serviço	43
Configurações avançadas	44
AWSSDK.Extensões. NETCore.Configuração e IConfiguration	45
Configuração de outros parâmetros do aplicativo	49
Referência dos arquivos de configuração para o AWS SDK para .NET	57
Uso de credenciais herdadas	70
Avisos e orientações importantes para credenciais	70
Usando o arquivo de AWS credenciais compartilhado	71
Usando a SDK Store (somente Windows)	75
Atributos do SDK	79
APIs assíncronas	79
Novas tentativas e tempos limite	81
Novas tentativas	81

Tempos limite	83
Paginadores	86
Onde posso encontrar paginadores?	86
O que os paginadores me oferecem?	86
Paginação síncrona em comparação à assíncrona	87
Exemplo	87
Considerações adicionais sobre paginadores	91
Observabilidade	92
Recursos adicionais do	92
Configurar um TelemetryProvider	92
Métricas	94
Provedores de telemetria	96
Ferramentas adicionais	97
Ferramenta de implantação da AWS	97
Estrutura de processamento de mensagens para .NET da AWS	98
Integrações ao .NET Aspire	98
Autenticação avançada	99
Autenticação única	99
Pré-requisitos	100
Como configurar um perfil de SSO	100
Como gerar e usar de tokens de SSO	102
Recursos adicionais	107
Tutoriais	107
Tutorial: somente aplicativo .NET	107
Tutorial: AWS CLI e aplicativo.NET	116
Implantar em AWS	126
Implantar a partir da CLI do .NET	126
Implante a partir dos kits de ferramentas IDE	126
Casos de uso	127
Aplicativos ASP.NET Core:	127
Aplicativos do .NET Console	128
Aplicativos do Blazor WebAssembly	129
AWS Lambda projetos	130
Pré-requisitos	130
Comandos do Lambda disponíveis	130
Etapas para implantar	131

Migre o seu projeto	133
Migrar para a versão 3	133
Sobre as versões do AWS SDK para .NET	133
Redefinição da arquitetura para o SDK	133
Alterações que podem causar interrupções	133
Migrar para a versão 3.5	135
O que mudou na versão 3.5	135
Migrar código síncrono	137
Migrar para a versão 3.7	138
Migração do .NET Standard 1.3	138
Trabalhe com AWS serviços	140
Exemplos de código com orientação	140
CloudFormation	141
Amazon Cognito	145
DynamoDB	153
Amazon EC2	183
IAM	246
Amazon S3	265
Amazon SNS	275
Amazon SQS	280
AWS Lambda	313
APIs	313
Pré-requisitos	313
Mais informações	313
Tópicos	313
Lambda Annotations	314
Bibliotecas e estruturas de alto nível	315
Estrutura de processamento de mensagens	316
Integrar a AWS ao .NET Aspire	336
Outros serviços e configuração	337
Exemplos de código	339
ACM	341
Ações	341
API Gateway	345
Cenários	346
AWS contribuições da comunidade	347

Aurora	347
Cenários	346
ajuste de escala automático	348
Ações	341
Cenários	346
Amazon Bedrock Runtime	400
Cenários	346
Amazon Nova	416
Amazon Nova Canvas	436
Claude da Anthropic	438
Command da Cohere	440
Llama da Meta	444
Mistral AI	448
CloudWatch Registros	451
Ações	341
Provedor de identidade do Amazon Cognito	466
Ações	341
Cenários	346
Amazon Comprehend	490
Ações	341
Cenários	346
Amazon DocumentDB	502
Exemplos sem servidor	502
DynamoDB	506
Ações	341
Cenários	346
Exemplos sem servidor	502
AWS contribuições da comunidade	347
Amazon EC2	573
Conceitos básicos	573
Ações	341
Cenários	346
Amazon ECS	696
Ações	341
Cenários	346
Elastic Load Balancing Versão 2	708

Ações	341
Cenários	346
EventBridge	766
Conceitos básicos	766
Conceitos básicos	573
Ações	341
EventBridge Agendador	806
Conceitos básicos	766
Ações	341
Cenários	346
Amazon Glacier	836
Conceitos básicos	766
Ações	341
AWS Glue	846
Conceitos básicos	766
Conceitos básicos	573
Ações	341
IAM	877
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Cenários	346
Amazon Keyspaces	974
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Kinesis	1002
Ações	341
Exemplos sem servidor	502
AWS KMS	1020
Ações	341
Lambda	1032
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Cenários	346

Exemplos sem servidor	502
AWS contribuições da comunidade	347
MediaConvert	1083
Conceitos básicos	766
Ações	341
Amazon MSK	1094
Exemplos sem servidor	502
Organizações	1095
Ações	341
Central de Parceiros	1114
Ações	341
Amazon Pinpoint	1120
Ações	341
Amazon Polly	1127
Ações	341
Cenários	346
Amazon RDS	1140
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Cenários	346
Exemplos sem servidor	502
Serviços de dados do Amazon RDS	1178
Cenários	346
Amazon Rekognition	1179
Ações	341
Cenários	346
Registro de domínios do Route 53	1211
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Amazon S3	1238
Ações	341
Cenários	346
Exemplos sem servidor	502
SageMaker IA	1380

Conceitos básicos	766
Ações	341
Cenários	346
Secrets Manager	1414
Ações	341
Amazon SES	1418
Ações	341
Cenários	346
API v2 do Amazon SES	1432
Ações	341
Cenários	346
Amazon SNS	1471
Conceitos básicos	766
Ações	341
Cenários	346
Exemplos sem servidor	502
Amazon SQS	1516
Conceitos básicos	766
Ações	341
Cenários	346
Exemplos sem servidor	502
Step Functions	1560
Conceitos básicos	766
Conceitos básicos	573
Ações	341
AWS STS	1588
Ações	341
Suporte	1591
Conceitos básicos	766
Conceitos básicos	573
Ações	341
Amazon Textract	1619
Cenários	346
Amazon Transcribe	1620
Ações	341
Amazon Translate	1632

Ações	341
Cenários	346
Segurança	1646
Proteção de dados	1647
Gerenciamento de Identidade e Acesso	1648
Público	1648
Autenticação com identidades	1649
Gerenciar o acesso usando políticas	1650
Como Serviços da AWS trabalhar com o IAM	1652
Solução de problemas AWS de identidade e acesso	1652
Validação de conformidade	1654
Resiliência	1655
Segurança da infraestrutura	1655
Aplicar uma versão mínima do TLS	1656
.NET Core	1656
NET Framework	1657
Ferramentas da AWS para PowerShell	1658
Xamarin	1659
Unity	1660
Navegador (para Blazor) WebAssembly	1660
Migração do cliente de criptografia S3 (V1 para V2)	1660
Visão geral da migração	1661
Atualize os clientes existentes para clientes V1 de transição para ler novos formatos.	1661
Migre clientes V1 de transição para clientes V2 para escrever novos formatos.	1662
Atualize os clientes V2 para que não leiam mais os formatos V1.	1666
Migração do cliente de criptografia S3 (V2 para V4)	1666
Compreendendo os conceitos da V4	1667
Atualize os clientes existentes para ler os formatos V4	1668
Migre para clientes V4	1669
Exemplos adicionais de configuração	1673
Considerações especiais	1676
Como obter assemblies do AWSSDK	1676
Faça download e extraia os arquivos ZIP	1676
Acessar credenciais e perfis em um aplicativo	1677
Exemplos para a aula CredentialProfileStoreChain	1678
Exemplos de aulas SharedCredentialsFile e AWSCredentials fábrica	1680

Suporte ao Unity	1680
Suporte do Xamarin	1681
Referência de API	1683
Sobre as versões de referência da API	1683
Histórico do documento	1686

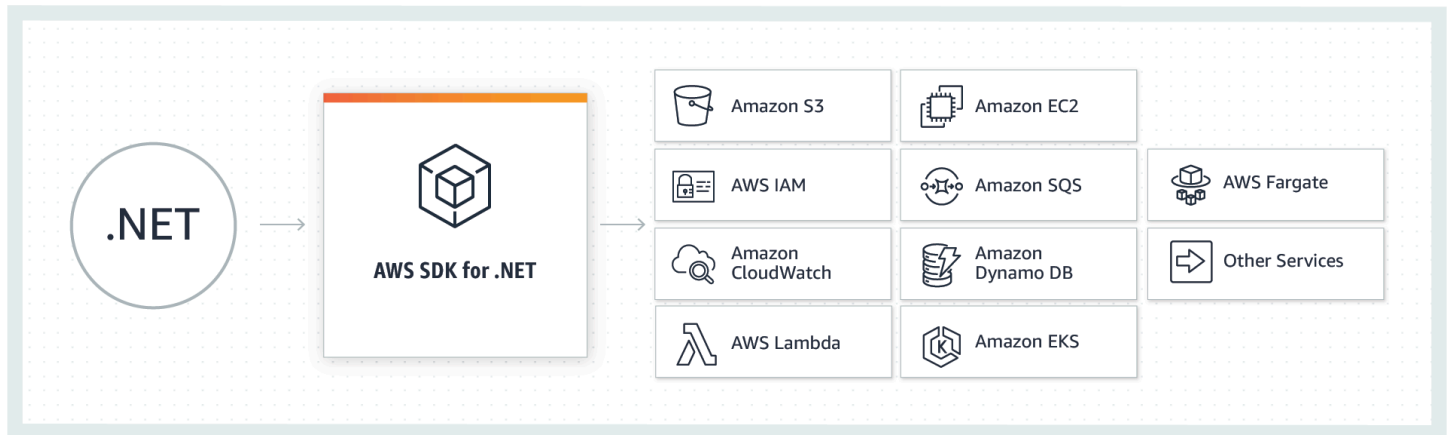
O AWS SDK para .NET V3 entrou no modo de manutenção.

Recomendamos que você migre para a [AWS SDK para .NET V4](#). Para obter detalhes e informações adicionais sobre como migrar, consulte nosso [anúncio do modo de manutenção](#).

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.

O que é o AWS SDK para .NET

O AWS SDK para .NET torna mais fácil criar aplicativos .NET que aproveitam os serviços de infraestrutura econômicos, dimensionáveis e confiáveis da AWS, como o Amazon Simple Storage Service (Amazon S3) e o Amazon Elastic Compute Cloud (Amazon EC2). O SDK simplifica o uso dos serviços da AWS fornecendo um conjunto de bibliotecas que são consistentes e familiares para os desenvolvedores .NET.



(OK, entendi! Está tudo pronto para [me preparar](#) e [fazer um tour rápido](#).)

Sobre essa versão

Documentação atual e anterior

Esta documentação é para a versão 3 (V3) do AWS SDK para .NET. A V3 não é a mais recente do SDK. Para a versão mais recente do SDK, consulte o [Guia do desenvolvedor do AWS SDK para .NET \(V4\)](#).

O foco principal desta documentação é o .NET Core e ASP.NET Core, mas também contém informações sobre o .NET Framework e o ASP.NET 4.x. Além do Windows e do Visual Studio, ele considera o desenvolvimento multiplataforma igualmente.

Para acessar informações sobre como realizar a migração de versões anteriores do SDK, consulte [Migre o seu projeto](#).

Para encontrar conteúdo obsoleto para versões anteriores do SDK, consulte os seguintes itens:

- [AWS SDK para .NET \(versão 2, obsoleta\) Guia do desenvolvedor do](#)

- [Referências de API obsoletas do AWS SDK para .NET](#)

Manutenção e suporte para as versões principais do SDK

Para obter informações sobre manutenção e suporte para versões principais do SDK e suas dependências subjacentes, consulte o seguinte no [Guia de referência de AWS SDKs e ferramentas](#):

- [AWS Política de manutenção de ferramentas e SDKs da](#)
- [AWS Matriz de suporte a versões de ferramentas e SDKs da](#)

Casos de uso comuns

O AWS SDK para .NET ajuda você a realizar vários casos de uso convincentes, incluindo os seguintes:

- Gerenciar usuários e funções com [AWS Identity and Access Management \(IAM\)](#).
- Acessar o [Amazon Simple Storage Service \(Amazon S3\)](#) para criar buckets e armazenar objetos.
- Gerenciar assinaturas de tópicos de HTTP do [Amazon Simple Notification Service \(Amazon SNS\)](#).
- Usar o [utilitário de transferência S3](#) para transferir arquivos de seus aplicativos Xamarin para o Amazon S3.
- Usar o [Amazon Simple Queue Service \(Amazon SQS\)](#) para processar mensagens e fluxos de trabalho entre componentes em um sistema.
- Executar transferências eficientes do Amazon S3 enviando instruções SQL para o [Amazon S3 Select](#).
- Criar e executar instâncias do [Amazon EC2](#), e configurar e solicitar [instâncias spot](#) do Amazon EC2.

Tópicos adicionais nesta seção

- [Ferramentas da AWS relacionadas ao AWS SDK para .NET](#)
- [Plataformas compatíveis com AWS SDK para .NET](#)
- [Guia de referência de AWS SDKs e ferramentas](#)

- [Histórico de revisões](#)
- [O que há de novo no AWS SDK para .NET](#)
- [Recursos adicionais](#)

Ferramentas da AWS relacionadas ao AWS SDK para .NET

Ferramentas para Windows PowerShell e ferramentas para PowerShell Core

O AWS Tools for Windows PowerShell e o AWS Tools for PowerShell Core são módulos do PowerShell criados com base na funcionalidade exposta pelo AWS SDK para .NET. As ferramentas para PowerShell da AWS permitem que você execute scripts para operações em seus recursos da AWS usando a linha de comando do PowerShell. Embora os cmdlets sejam implementados usando os clientes de serviços e os métodos do SDK, os cmdlets proporcionam uma experiência idiomática do PowerShell para especificar parâmetros e lidar com os resultados.

Para começar, consulte o [AWS Tools for Windows PowerShell](#).

Toolkit for VS Code

O [AWS Toolkit for Visual Studio Code](#) é um plug-in para o editor Visual Studio Code (VS Code). O toolkit facilita o desenvolvimento, a depuração e a implantação de aplicativos que usam a AWS.

Com o toolkit, é possível fazer o seguinte:

- Criar aplicativos sem servidor que contenham funções do AWS Lambda e implantá-los em uma pilha do AWS CloudFormation.
- Trabalhe com esquemas do Amazon EventBridge.
- Usar o IntelliSense ao trabalhar com arquivos de definição de tarefa do Amazon ECS.
- Visualizar um aplicativo do AWS Cloud Development Kit (AWS CDK).

Toolkit para Visual Studio

O AWS Toolkit for Visual Studio é um plug-in do Visual Studio IDE que torna mais fácil desenvolver, depurar e implantar aplicativos .NET que usam Amazon Web Services. O Toolkit para Visual Studio

fornece modelos do Visual Studio para serviços, como o Lambda, e assistentes de implantação para aplicativos web e aplicativos sem servidor. Você pode usar o AWS Explorer para gerenciar instâncias do Amazon EC2, trabalhar com tabelas do Amazon DynamoDB, publicar mensagens para as filas do Amazon Simple Notification Service (Amazon SNS) e muito mais, tudo no Visual Studio.

Para começar, consulte [Setting up the AWS Toolkit for Visual Studio](#).

Toolkit for Azure DevOps

O Kit de ferramentas da AWS para Microsoft Azure DevOps adiciona tarefas para habilitar facilmente pipelines de compilação e lançamento no Azure DevOps e no Azure DevOps Server para trabalhar com serviços da AWS. Você pode trabalhar com Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, Lambda, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) e Amazon SNS. Também é possível executar comandos usando o módulo do Windows PowerShell e a AWS Command Line Interface (AWS CLI).

Para começar a usar o Kit de ferramentas da AWS para Azure DevOps, consulte o [Guia do Usuário do Kit de ferramentas da AWS para Microsoft Azure DevOps](#).

Plataformas compatíveis com AWS SDK para .NET

O AWS SDK para .NET fornece grupos distintos de conjuntos para desenvolvedores direcionarem a diferentes plataformas. Contudo, nem toda funcionalidade do SDK é a mesma em cada uma dessas plataformas. Este tópico descreve as diferenças no suporte para cada plataforma.

.NET Core

O AWS SDK para .NET oferece suporte a aplicativos escritos para o .NET Core (.NET Core 3.1, .NET 5, .NET 6 e assim por diante). Os clientes de serviço da AWS oferecem suporte somente a padrões de chamadas assíncronas no .NET core. Isso também afeta várias das abstrações de alto nível criadas sobre clientes de serviços, como o `TransferUtility` do Amazon S3, que só oferece suporte a chamadas assíncronas no ambiente .NET Core.

.NET Standard 2.0

Variações não estruturais do AWS SDK para .NET estão em conformidade com o [.NET Standard 2.0](#). O AWS SDK para .NET fornece somente métodos assíncronos para aplicativos escritos com o .NET Standard.

.NET Framework 4.5

Warning

A partir de 15 de agosto de 2024, o AWS SDK para .NET encerrará o suporte para o .NET Framework 3.5 e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

Essa versão do AWS SDK para .NET é compilada com o .NET Framework 4.5 e é executada no runtime do .NET 4.0. Os clientes de serviço da AWS oferecem suporte a padrões de chamada síncrona e assíncrona, e usam as palavras-chave [async e await](#) introduzidas no [C# 5.0](#).

.NET Framework 3.5

Warning

A partir de 15 de agosto de 2024, o AWS SDK para .NET encerrará o suporte para o .NET Framework 3.5 e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

Essa versão do AWS SDK para .NET é compilada com o .NET Framework 3.5 e é executada tanto no runtime do .NET 2.0 quanto do .NET 4.0. Os clientes de serviço da AWS oferecem suporte a padrões de chamada síncrona e assíncrona, e usam os padrões Begin e End.

Note

O AWS SDK para .NET não é compatível com FIPS (Federal Information Processing Standard) quando usado por aplicativos criados na versão 2.0 do CLR. Para obter detalhes sobre como você pode substituir uma implementação compatível com FIPS nesse ambiente, consulte [CryptoConfig](#) no blog da Microsoft e a classe HMACSHA256 da equipe de [segurança de CLR](#) (HMACSHA256Cng) em Security.Cryptography.dll.

Biblioteca de classes portátil e Xamarin

O AWS SDK para .NET também contém uma implementação de biblioteca de classes portátil. A implementação da biblioteca de classes portátil pode ser destinada a várias plataformas, incluindo Universal Windows Platform (UWP) e Xamarin em iOS e Android. Consulte [Mobile SDK for .NET e Xamarin](#) para obter mais detalhes. Os clientes de serviço da AWS oferecem suporte somente a padrões de chamada assíncrona.

Suporte ao Unity

Para obter informações sobre o suporte do Unity, consulte [Considerações especiais sobre o suporte ao Unity](#).

Mais informações

[Migrar para a versão 3.5 do AWS SDK para .NET](#)

Guia de referência de AWS SDKs e ferramentas

O [Guia de referência de SDKs e ferramentas da AWS](#) contém informações relevantes e importantes para muitos dos SDKs e kits de ferramentas da AWS e para AWS CLI. Veja a seguir alguns exemplos das informações que a referência contém:

- Informações sobre os [arquivos compartilhados config e credentials da AWS](#) e a [localização](#) deles.
- [Configurando contas, usuários e perfis da AWS](#)
- [Referência de configurações e definições de autenticação](#)
- [Bibliotecas do Common Runtime \(CRT\) da AWS](#)
- [AWS Política de manutenção de ferramentas e SDKs da](#)
- [AWS Matriz de suporte a versões de ferramentas e SDKs da](#)

Histórico de revisões

Para descobrir o que mudou em várias versões, veja o seguinte:

- [Logs de alterações do SDK](#)

- [O que há de novo no AWS SDK para .NET](#)
- [Histórico do documento](#)

O que há de novo no AWS SDK para .NET

Para acessar informações gerais sobre novos desenvolvimentos relacionados ao AWS SDK para .NET, consulte a página do produto em <https://aws.amazon.com/sdk-for-net/> e os [logs de alteração de SDK](#).

Veja a seguir o que há de novo no AWS SDK para .NET.

17 de outubro de 2025: versão 2.0 do AWS Deploy Tool

Foi lançada a versão 2.0 da CLI da ferramenta de implantação da AWS para .NET. Para acessar informações sobre o Deploy Tool, consulte [Implantar aplicativos na AWS](#). Para acessar informações sobre a versão 2.0 do Deploy Tool, consulte a postagem do blog [What's New in the AWS Deploy Tool for .NET](#).

21 de agosto de 2025: fim do suporte para a versão 3 do AWS SDK para .NET em breve

Foi anunciado o fim do suporte para a versão (V3) do AWS SDK para .NET. Consulte o [Guia de migração da V4](#) para migrar suas aplicações V3 e evitar interrupções. Para acessar mais informações, consulte a postagem do blog [Announcing the end-of-support for the AWS SDK para .NET v3](#).

5 de maio de 2025: disponibilidade geral do AWS Message Processing Framework para .NET.

O [AWS Message Processing Framework for .NET](#) está disponível para o público! Trata-se de um framework nativo da AWS que simplifica o desenvolvimento de aplicações de processamento de mensagens .NET que usam serviços da AWS, como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon EventBridge.

28 de abril de 2025: versão 4 do AWS SDK para .NET

A versão 4 do AWS SDK para .NET está disponível ao público! Para acessar mais informações, consulte o [Guia do desenvolvedor do AWS SDK para .NET \(V4\)](#), especialmente o tópico sobre [migração para a versão 4](#).

15 de fevereiro de 2025: integrações ao .NET Aspire.

Foram lançadas integrações ao .NET Aspire para melhorar o ciclo interno de desenvolvimento. Para mais informações, consulte [Integrar a AWS ao .NET Aspire no AWS SDK para .NET](#).

10 de fevereiro de 2025: versão de disponibilidade geral do recurso observabilidade.

Observabilidade é a medida em que o estado atual de um sistema pode ser identificado com base nos dados que ele emite. Esse recurso foi adicionado ao AWS SDK para .NET, incluindo a implementação de um provedor de telemetria. Para acessar mais informações, consulte [Observabilidade](#) neste guia e a postagem do blog [Announcing the general availability of AWS .NET OpenTelemetry libraries](#).

15 de janeiro de 2025: novo comportamento padrão da proteção de integridade.

A partir da versão 3.7.412.0 do AWS SDK para .NET, o SDK fornece proteções de integridade padrão calculando automaticamente uma soma de verificação CRC32 para uploads. Para acessar mais informações, consulte o anúncio no GitHub em <https://github.com/aws/aws-sdk-net/issues/3610>. O SDK também fornece configurações globais para proteções de integridade de dados que você pode definir externamente. Leia mais sobre elas em [Data Integrity Protections](#) no [Guia de referência de AWS SDKs e ferramentas](#).

15 de novembro de 2024: publicação do blog da versão prévia 4 da versão 4.

A versão prévia 4 do AWS SDK para .NET versão 4 foi lançada em 15 de novembro de 2024. Para acessar mais informações sobre essa versão prévia, consulte a postagem do blog [Preview 4 of AWS SDK para .NET V4](#).

16 de agosto de 2024: postagem do blog da versão prévia 1 da versão 4.

O AWS SDK para .NET versão 4 foi lançada como primeira versão prévia em 16 de agosto de 2024. Para acessar mais informações sobre essa versão prévia, consulte a postagem do blog [Preview 1 of AWS SDK para .NET V4](#).

23 de fevereiro de 2024: adição de suporte para o .NET 8.

Foi adicionado suporte para .NET 8 ao AWS SDK para .NET. Use os [pacotes NuGet](#) mais recentes ou as [montagens que comportam o .NET 8 e posterior](#). Acesse informações adicionais sobre esse suporte, incluindo [suporte para Lambda](#) na postagem do blog [.NET 8 Support on AWS](#).

18 de fevereiro de 2024: alterações no suporte do .NET Framework em breve.

A partir de 15 de agosto de 2024, o AWS SDK para .NET encerrará o suporte para o .NET Framework 3.5 e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais

informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

17-07-2023: A estrutura AWS Lambda Annotations foi lançada para disponibilidade geral.

A [estrutura AWS Lambda Annotations](#) faz com que a experiência de escrever funções do Lambda em C# pareça mais natural para desenvolvedores .NET usando a tecnologia de geração de código-fonte C#. Ela agora está disponível ao público em geral.

2023-07-15: O provedor de cache distribuído para DynamoDB foi lançado em versão de pré-visualização

A biblioteca Distributed Cache Provider permite que o Amazon DynamoDB seja usado como armazenamento para a estrutura de cache distribuído do ASP.NET Core. Para obter mais informações, consulte a postagem do blog [Introducing the AWS .NET Distributed Cache Provider for DynamoDB \(Preview\)](#) e o [repositório GitHub](#).

13-07-2022: A ferramenta de Implantação da AWS foi lançada

A ferramenta de Implantação da AWS foi lançada. Ela é uma ferramenta interativa para a CLI do .NET e o AWS Toolkit for Visual Studio que ajuda a implantar aplicativos .NET com conhecimento mínimo sobre a AWS e com o menos cliques ou comandos. Para obter mais informações, consulte [Implantar aplicativos na AWS](#).

2020-08-24: A versão 3.5 do SDK foi lançada

- Padronizou ainda mais a experiência do .NET fazendo a transição do suporte para todas as variações que não sejam do Framework do SDK para o .NET Standard 2.0. Consulte [Migrar para a versão 3.5](#) para obter mais informações.
- Foram adicionados paginadores a muitos clientes de serviços, o que torna a paginação dos resultados da API mais conveniente. Para obter mais informações, consulte [Paginadores](#).

Recursos adicionais

Serviços com suporte

O AWS SDK para .NET oferece suporte à maioria dos produtos de infraestrutura da AWS e mais serviços são adicionados com frequência. Para obter uma lista de serviços da AWS compatíveis com o SDK, consulte o [arquivo SDK README](#).

Página inicial do AWS SDK para .NET

Para obter mais informações sobre o AWS SDK para .NET, acesse a página inicial do SDK em <https://aws.amazon.com/sdk-for-net/>.

Documentação de referência do SDK

A documentação de referência do SDK dá a você a capacidade de pesquisar e buscar todos os códigos incluídos no SDK. Ele fornece documentação completa e exemplos de uso. Para obter mais informações, consulte a [Referência da API do AWS SDK para .NET](#).

AWS re:Post (antigos AWS forums)

Visite o [AWS re:Post](#), especificamente o [tópico sobre o AWS SDK para .NET](#), para tirar dúvidas ou fornecer feedback sobre a AWS. Cada página de documentação tem um link Experimente o AWS re:post na parte inferior da página que leva você ao tópico re:POST associado. Engenheiros da AWS monitoram os tópicos e respondem a perguntas, feedback e problemas.

Se você estiver conectado ao re:POST, você também pode seguir um tópico. Para seguir o tópico para o AWS SDK para .NET, acesse a página [Todos os tópicos](#), encontre ".NET em AWS" e selecione o botão Seguir.

Toolkits

- AWS Toolkit for Visual Studio: Se usar o Microsoft Visual Studio IDE, você deverá verificar o [Guia do usuário do AWS Toolkit for Visual Studio](#).
- AWS Toolkit for Visual Studio Code: Se usar o Microsoft Visual Studio IDE, você deverá verificar o [Guia do usuário do AWS Toolkit for Visual Studio Code](#).

Extensões, bibliotecas e ferramentas úteis

Acesse os repositórios [aws/dotnet](#) e [aws/aws-sdk-net](#) no site do Github para obter links para bibliotecas, ferramentas e recursos que você pode usar para ajudar a criar serviços e aplicativos.NET na AWS.

Veja os seguintes exemplos:

- [AWS .Extensão de configuração do AWS.NET para Systems Manager da](#)
- [AWS Configuração do .NET Core para extensões da](#)
- [AWS Registro em log .NET da](#)

- [Biblioteca de extensão de autenticação do Amazon Cognito](#)
- [AWS X-Ray SDK for .NET](#)

Outros recursos

A seguir estão outros recursos que podem ser úteis:

- [Rede de desenvolvedores](#)
- [Ambiente de desenvolvimento.NET na nuvem da AWS - Implantação de referência de início rápido](#)
- [Blog Hello, Cloud](#)
- AWSWhitepaper: [Como desenvolver e implantar aplicativos.NET na AWS](#)
- [AWS Microservice Extractor for .NET](#)
- [Assistente de portabilidade para .NET](#)
- [AWS Guia de referência de SDKs e ferramentas](#)

Conceitos básicos da AWS SDK para .NET

Para usar o AWS SDK para .NET, você precisa instalar sua cadeia de ferramentas e configurar várias coisas essenciais que seu aplicativo precisa para acessar os serviços da AWS. Isso inclui:

- Uma conta ou perfil de usuário apropriado
- Informações de autenticação para essa conta de usuário ou para assumir esse perfil
- Especificação da região da AWS
- Pacotes ou assemblies do AWSSDK

Alguns dos tópicos desta seção fornecem informações sobre como configurar esses itens essenciais.

Outros tópicos desta e de outras seções fornecem informações sobre formas avançadas de configurar o seu projeto.

Tópicos

- [Instale e configure seu conjunto de ferramentas](#)
- [Configure a autenticação do SDK com AWS](#)
- [Faça um tour rápido pelo AWS SDK para .NET](#)
- [Iniciar um novo projeto](#)
- [Configurar a região da AWS](#)
- [Instale pacotes do AWSSDK com o NuGet](#)
- [Instale assemblies do AWSSDK sem o NuGet](#)
- [Resolução de perfil e credenciais](#)
- [Informações adicionais sobre usuários e perfis](#)
- [Configurações avançadas para seu projeto do AWS SDK para .NET](#)
- [Uso de credenciais herdadas](#)

Instale e configure seu conjunto de ferramentas

Para usar o AWS SDK para .NET, é necessário ter determinadas ferramentas de desenvolvimento instaladas.

Desenvolvimento entre plataformas

Os itens a seguir são necessários para desenvolvimento .NET para várias plataformas no Windows, no Linux ou no macOS:

- Microsoft [.NET Core SDK](#), versão 2.1, 3.1 ou posterior, que inclua a interface de linha de comando (CLI) do .NET (**dotnet**) e o tempo de execução do .NET Core.
- Um editor de código ou ambiente de desenvolvimento integrado (IDE) apropriado para seu sistema operacional e requisitos. Normalmente, ele fornece algum suporte para .NET Core.

Exemplos incluem [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) e [Microsoft Visual Studio](#).

- (Opcional) Um kit de ferramentas da AWS, se houver algum disponível, para o editor que você escolheu e seu sistema operacional.

Os exemplos incluem [AWS Toolkit for Visual Studio Code](#), [AWS Toolkit for JetBrains](#) e [AWS Toolkit for Visual Studio](#).

Windows com Visual Studio e .NET Core

Os itens a seguir são necessários para desenvolvimento no Windows com Visual Studio e .NET Core:

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

Isso geralmente é incluído por padrão ao instalar uma versão recente do Visual Studio.

- (Opcional) O AWS Toolkit for Visual Studio, que é um plug-in que fornece uma interface de usuário para gerenciar seus recursos da AWS e perfis locais no Visual Studio. Para instalar o toolkit, consulte [Setting up the AWS Toolkit for Visual Studio](#).

Para obter mais informações, consulte o [Guia do usuário do AWS Toolkit for Visual Studio](#).

Próxima etapa

[Configure a autenticação do SDK com AWS](#)

Configure a autenticação do SDK com AWS

Você deve estabelecer como seu código é autenticado AWS ao desenvolver com Serviços da AWS. Há diferentes maneiras de configurar o acesso programático aos AWS recursos, dependendo do ambiente e do AWS acesso disponível para você.

Para ver vários métodos de autenticação para o SDK, consulte [Autenticação e acesso](#) no Guia de referência de ferramentas AWS SDKs e ferramentas.

Este tópico pressupõe que um novo usuário esteja se desenvolvendo localmente, não tenha recebido um método de autenticação do empregador e o usará Centro de Identidade do AWS IAM para obter credenciais temporárias. Se seu ambiente não se enquadra nessas suposições, algumas das informações neste tópico podem não se aplicar a você ou algumas das informações podem já ter sido fornecidas.

A configuração desse ambiente requer várias etapas, que são resumidas da seguinte forma:

1. [Faça login usando as credenciais do console](#)
2. [Habilitar e configurar o Centro de Identidade do IAM](#)
3. [Configure o SDK para usar o Centro de Identidade do IAM.](#)
4. [Iniciar uma sessão do portal de AWS acesso](#)

Faça login usando as credenciais do console

Você pode usar suas credenciais de login existentes do AWS Management Console para acesso programático aos serviços. AWS Depois de um fluxo de autenticação baseado em navegador, AWS gera credenciais temporárias que funcionam em ferramentas de desenvolvimento locais, como a AWS CLI e o Ferramentas da AWS para PowerShell SDK para.NET v3. AWS

Com esse processo, você pode se autenticar usando credenciais raiz criadas durante a configuração inicial da conta, um usuário do IAM ou uma identidade federada do seu provedor de identidade, e a AWS CLI ou gerencia Ferramentas da AWS para PowerShell automaticamente as credenciais temporárias para você. Essa abordagem aumenta a segurança ao eliminar a necessidade de armazenar credenciais de longo prazo localmente.

Ao executar o `aws login` comando da AWS CLI, você pode selecionar entre suas sessões ativas do console ou entrar por meio do fluxo de autenticação baseado em navegador e isso gerará automaticamente credenciais temporárias. O token de credencial temporário expirará em 15 minutos, mas o AWS SDK para.NET v3 atualizará automaticamente o token quando necessário durante suas solicitações.

 Warning

Esse recurso não é compatível com destinos do .NET Framework.

 Important

Se você estiver usando as credenciais do console para autenticação, seu aplicativo deverá fazer referência aos seguintes NuGet pacotes para que a resolução de AWS login funcione:

- AWSSDK.Fazer login

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

- [Faça login usando credenciais do console usando AWS CLI.](#)

O config arquivo ficaria assim que você fizesse o login seguindo as etapas fornecidas acima na 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
```

Habilitar e configurar o Centro de Identidade do IAM

Para usar o Centro de Identidade do IAM, primeiro ele deve ser habilitado e configurado. Para ver detalhes sobre como fazer isso com o SDK, consulte a Etapa 1 no tópico sobre [autenticação do IAM](#)

[Identity Center](#) no Guia de referência de ferramentas AWS SDKs e ferramentas. Especificamente, siga todas as instruções necessárias em Não estabeleci acesso por meio do Centro de Identidade do IAM.

Configure o SDK para usar o Centro de Identidade do IAM.

As informações sobre como configurar o SDK para usar o IAM Identity Center estão na Etapa 2 do tópico sobre [autenticação do IAM Identity Center](#) no Guia de referência de ferramentas AWS SDKs e ferramentas. Depois de concluir essa configuração, o sistema deverá conter os seguintes elementos:

- O AWS CLI, que você usa para iniciar uma sessão do portal de AWS acesso antes de executar seu aplicativo.
- O AWS config arquivo compartilhado que contém um [\[default\]perfil](#) com um conjunto de valores de configuração que podem ser referenciados no SDK. Para encontrar a localização desse arquivo, consulte [Localização dos arquivos compartilhados no](#) Guia de referência de ferramentas AWS SDKs e ferramentas. O AWS SDK para .NET usa o provedor de token SSO do perfil para adquirir credenciais antes de enviar solicitações para. AWS O valor `sso_role_name`, que é um perfil do IAM conectado a um conjunto de permissões do Centro de Identidade do IAM, deve permitir o acesso a Serviços da AWS usado na aplicação.

O arquivo de exemplo config a seguir mostra um perfil padrão configurado com o provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção chamada `sso-session`. A `sso-session` seção contém configurações para iniciar uma sessão do portal de AWS acesso.

```
[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

Se você estiver usando Centro de Identidade do AWS IAM para autenticação, seu aplicativo deverá fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:

- AWSSDK.SSO
- AWSSDK.SSO0IDC

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

Iniciar uma sessão do portal de AWS acesso

Antes de executar um aplicativo que acessa Serviços da AWS, você precisa de uma sessão ativa do portal de AWS acesso para que o SDK use a autenticação do IAM Identity Center para resolver as credenciais. Dependendo da duração da sessão configurada, o seu acesso acabará expirando e o SDK encontrará um erro de autenticação. Para entrar no portal de AWS acesso, execute o seguinte comando no AWS CLI.

```
aws sso login
```

Como você tem uma configuração de perfil padrão, não precisa chamar o comando com uma opção `--profile`. Se a configuração do provedor de token de SSO estiver usando um perfil nomeado, o comando será `aws sso login --profile named-profile`.

Para testar se você já tem uma sessão ativa, execute o AWS CLI comando a seguir.

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

A resposta a esse comando deve relatar a conta do Centro de Identidade do IAM e o conjunto de permissões configurados no arquivo compartilhado `config`.

 Note

Se você já tiver uma sessão ativa do portal de AWS acesso e executá-la `aws sso login`, não será necessário fornecer credenciais.

O processo de login pode solicitar que você permita o AWS CLI acesso aos seus dados. Como o AWS CLI é construído sobre o SDK para Python, as mensagens de permissão podem conter variações do botocore nome.

Mais informações

- Para obter informações adicionais sobre como usar o IAM Identity Center e o SSO em um ambiente de desenvolvimento, consulte [Autenticação única](#) na seção [Autenticação avançada](#). Essas informações incluem métodos alternativos e mais avançados, bem como tutoriais que mostram como usar esses métodos.
- Para obter mais opções de autenticação para o SDK, como o uso de perfis e variáveis de ambiente, consulte o capítulo de [configuração](#) no Guia de referência de ferramentas AWS SDKs e ferramentas.
- Para saber mais sobre as práticas recomendadas, consulte [Práticas recomendadas de segurança no IAM](#) no Guia do usuário do IAM.
- Para criar AWS credenciais de curto prazo, consulte [Credenciais de segurança temporárias](#) no Guia do usuário do IAM.
- Para saber mais sobre outros provedores de credenciais, consulte Provedores de [credenciais padronizados no Guia de](#) Referência de Ferramentas AWS SDKs e Ferramentas.

Faça um tour rápido pelo AWS SDK para .NET

Essa seção fornece tutoriais básicos para desenvolvedores que não estão familiarizados com o AWS SDK para .NET.

Note

Antes de usar esses tutoriais, primeiro você deve ter [instalado seu conjunto de ferramentas](#) e [configurado a autenticação do SDK](#).

Consulte informações sobre como desenvolver software para serviços da AWS específicos com exemplos de código em [Trabalhe com AWS serviços](#). Consulte mais exemplos de código em [SDK para .NET exemplos de código](#).

Tópicos

- [Aplicativo simples para várias plataformas usando o AWS SDK para .NET](#)
- [Aplicativo simples baseado no Windows usando o AWS SDK para .NET](#)
- [Próximas etapas](#)

Aplicativo simples para várias plataformas usando o AWS SDK para .NET

Esse tutorial usa o AWS SDK para .NET e o .NET Core para o desenvolvimento entre plataformas. O tutorial mostra como usar o SDK para listar os [buckets do Amazon S3](#) que você possui e criar um bucket de forma opcional.

Você executará este tutorial usando ferramentas para várias plataformas, como a interface de linha de comando (CLI) do .NET. Para outras maneiras de configurar seu ambiente de desenvolvimento, consulte [Instale e configure seu conjunto de ferramentas](#).

Necessário para desenvolvimento .NET para várias plataformas no Windows, no Linux ou no macOS:

- Microsoft [.NET Core SDK](#), versão 2.1, 3.1 ou posterior, que inclua a interface de linha de comando (CLI) do .NET (**dotnet**) e o tempo de execução do .NET Core.
- Um editor de código ou ambiente de desenvolvimento integrado (IDE) apropriado para seu sistema operacional e requisitos. Normalmente, ele fornece algum suporte para .NET Core.

Exemplos incluem [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) e [Microsoft Visual Studio](#).

Note

Antes de usar esses tutoriais, primeiro você deve ter [instalado seu conjunto de ferramentas](#) e [configurado a autenticação do SDK](#).

Etapas

- [Criar o projeto](#)
- [Criar o código](#)

- [Execute o aplicativo](#)
- [Limpeza](#)

Criar o projeto

1. Abra o prompt de comando ou terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.
2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

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

3. Vá para a pasta `S3CreateAndList` recém-criada e execute os comandos a seguir.

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

O comando anterior instala os pacotes NuGet do [gerenciador de pacotes NuGet](#). Como sabemos exatamente quais pacotes NuGet precisamos para esse tutorial, podemos executar esta etapa agora. Também é bastante comum conhecer os pacotes necessários durante o desenvolvimento. Quando isso acontece, um comando semelhante pode ser executado nesse momento.

Criar o código

1. Na pasta `S3CreateAndList`, localize e abra `Program.cs` no editor de código.
2. Substitua o conteúdo pelo código a seguir e salve o arquivo.

```
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;
    }
}
}

```

Execute o aplicativo

1. Execute o comando a seguir.

```
dotnet run
```

2. Examine a saída para ver o número de buckets do Amazon S3 que você possui e seus nomes, se houver.
3. Escolha um nome para um novo bucket do Amazon S3. Use "dotnet-quicktour-s3-1-cross-" como base e adicione algo exclusivo a ele, como um GUID ou seu nome. Siga as regras para nomes de bucket descritas em [Regras de nomeação de bucket](#) no [Guia do usuário do Amazon S3](#).
4. Execute o comando a seguir, substituindo *amzn-s3-demo-bucket* pelo nome do bucket escolhido.

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

5. Examine a saída para ver o novo bucket que foi criado.

Limpeza

Ao executar este tutorial, você criou alguns recursos que pode escolher para limpar neste momento.

- Se você não quiser manter o bucket que o aplicativo criou em uma etapa anterior, o exclua usando o console do Amazon S3 em <https://console.aws.amazon.com/s3/>.
- Se você não quiser manter seu projeto .NET, remova a pasta S3CreateAndList do seu ambiente de desenvolvimento.

Para onde ir em seguida

Volte para o [menu do tour rápido](#) ou vá direto para o [final deste tour rápido](#).

Aplicativo simples baseado no Windows usando o AWS SDK para .NET

Este tutorial usa o AWS SDK para .NET no Windows com o Visual Studio e o .NET Core. O tutorial mostra como usar o SDK para listar os [buckets do Amazon S3](#) que você possui e criar um bucket de forma opcional.

Você executará este tutorial no Windows usando o Visual Studio e o .NET Core. Para outras maneiras de configurar seu ambiente de desenvolvimento, consulte [Instale e configure seu conjunto de ferramentas](#).

Necessário para desenvolvimento no Windows com o Visual Studio e o .NET Core:

- [Microsoft Visual Studio](#)

- Microsoft .NET Core 2.1, 3.1 ou posterior

Isso geralmente é incluído por padrão ao instalar uma versão recente do Visual Studio.

Note

Antes de usar esses tutoriais, primeiro você deve ter [instalado seu conjunto de ferramentas](#) e [configurado a autenticação do SDK](#).

Etapas

- [Criar o projeto](#)
- [Criar o código](#)
- [Execute o aplicativo](#)
- [Limpeza](#)

Criar o projeto

1. Abra o Visual Studio e crie um projeto que use a versão C# do modelo de aplicação de console; ou seja, com a descrição: "...para criar uma aplicação de linha de comandos que pode ser executada no .NET Core..." Nomeie o projeto S3CreateAndList.

Note

Não escolha a versão do .NET Framework do modelo de aplicativo de console ou, se escolher, certifique-se de usar o .NET Framework 4.7.2 ou posterior.

2. Com o projeto recém-criado carregado, selecione Ferramentas, Gerenciador de pacotes NuGet, Gerenciar pacotes NuGet para a solução.
3. Procure os seguintes pacotes NuGet e instale-os no projeto: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO e AWSSDK.SSO0IDC

Esse processo instala o pacote NuGet do [gerenciador de pacotes NuGet](#). Como sabemos exatamente quais pacotes NuGet precisamos para esse tutorial, podemos executar esta etapa agora. Também é bastante comum conhecer os pacotes necessários durante o

desenvolvimento. Quando isso acontecer, siga um processo semelhante para instalá-los nesse momento.

4. Se você pretende executar o aplicativo no prompt de comando, abra um prompt de comando agora e navegue até a pasta que conterá a saída de compilação. Normalmente, isso é algo como `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, mas dependerá do seu ambiente.

Criar o código

1. No projeto `S3CreateAndList`, localize e abra `Program.cs` no IDE.
2. Substitua o conteúdo pelo código a seguir e salve o arquivo.

```
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;  
    }  
}  
}
```

3. Compile o aplicativo.

Note

Se estiver usando uma versão mais antiga do Visual Studio, você pode receber um erro de compilação semelhante ao seguinte:

“O atributo 'async main' não está disponível no C# 7.0. Use a versão 7.1 ou superior da linguagem.”

Se você receber esse erro, configure seu projeto para usar uma versão posterior da linguagem. Isso normalmente é feito nas propriedades do projeto, Build, Advanced.

Execute o aplicativo

1. Execute o aplicativo sem argumentos de linha de comando. Faça isso no prompt de comando (se você abriu um anteriormente) ou no IDE.
2. Examine a saída para ver o número de buckets do Amazon S3 que você possui e seus nomes, se houver.
3. Escolha um nome para um novo bucket do Amazon S3. Use "dotnet-quicktour-s3-1-winvs-" como base e adicione algo exclusivo a ele, como um GUID ou seu nome. Siga as regras para nomes de bucket descritas em [Regras de nomeação de bucket](#) no [Guia do usuário do Amazon S3](#).
4. Execute o aplicativo novamente, desta vez fornecendo o nome do bucket.

Na linha de comandos, substitua *amzn-s3-demo-bucket* no comando a seguir pelo nome do bucket escolhido.

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

Ou, se você estiver executando o aplicativo no IDE, selecione Projeto, Propriedades S3CreateAndList, Depurar e insira o nome do bucket ali.

5. Examine a saída para ver o novo bucket que foi criado.

Limpeza

Ao executar este tutorial, você criou alguns recursos que pode escolher para limpar neste momento.

- Se você não quiser manter o bucket que o aplicativo criou em uma etapa anterior, o exclua usando o console do Amazon S3 em <https://console.aws.amazon.com/s3/>.
- Se você não quiser manter seu projeto .NET, remova a pasta S3CreateAndList do seu ambiente de desenvolvimento.

Para onde ir em seguida

Volte para o [menu do tour rápido](#) ou vá direto para o [final deste tour rápido](#).

Próximas etapas

Certifique-se de limpar todos os recursos criados durante a execução desses tutoriais. Eles podem ser recursos da AWS ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Agora que você conheceu o AWS SDK para .NET, talvez queira [começar seu projeto](#).

Iniciar um novo projeto

Há várias técnicas que podem ser usadas para iniciar um novo projeto para acessar os serviços da AWS. A seguir estão alguns desses métodos:

- Se você for novo no desenvolvimento do .NET na AWS ou, pelo menos, novo no AWS SDK para .NET, poderá ver exemplos completos em [Faça um tour rápido](#). Ele fornece uma introdução ao SDK.

- É possível iniciar um projeto básico usando a CLI do .NET. Para ver um exemplo disso, abra um prompt de comando ou terminal, crie uma pasta ou diretório e navegue até ele e, em seguida, insira o seguinte.

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

Será criado um projeto vazio, ao qual você poderá adicionar código e pacotes NuGet. Para obter mais informações, consulte o [Guia do .NET Core](#).

Para ver uma lista de modelos de projeto, use o seguinte: `dotnet new --list`

- O AWS Toolkit for Visual Studio inclui modelos de projeto em C# para uma variedade de serviços da AWS. Depois de [instalar o toolkit](#) no Visual Studio, você poderá acessar os modelos ao criar um projeto.

Para ver isso, acesse [Como trabalhar com serviços da AWS](#) no [Guia do Usuário do AWS Toolkit for Visual Studio](#). Diversos exemplos nessa seção criam projetos.

- Se você desenvolve com o Visual Studio no Windows, mas sem o AWS Toolkit for Visual Studio, use suas técnicas habituais para criar um novo projeto.

Para ver um exemplo, abra o Visual Studio e selecione Arquivo, Novo, Projeto. Pesquise por “.net core” e escolha a versão C# do modelo Console App (.NET Core) ou WPF App (.NET Core). Será criado um projeto vazio, ao qual você poderá adicionar código e pacotes NuGet.

Você pode encontrar alguns exemplos de como trabalhar com serviços da AWS em [Exemplos de código com orientação](#).

Important

Se estiver usando Centro de Identidade do AWS IAM para autenticação, seu aplicativo deve fazer referência aos seguintes pacotes NuGet para que a resolução de SSO possa funcionar:

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

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

Configurar a região da AWS

As regiões da AWS permitem o acesso aos serviços da AWS que residem fisicamente em uma região geográfica específica. Isso pode ser útil para redundância e para manter os seus dados e aplicativos em execução próximo ao lugar em que você e os seus usuários os acessarão.

Para exibir a lista atual com todas as Regiões e endpoints compatíveis para cada serviço da AWS, consulte [Endpoints e cotas de serviço](#) em Referência geral da AWS. Para ver uma lista dos endpoints regionais existentes, consulte [endpoints de serviço da AWS](#). Para ver informações detalhadas sobre regiões, consulte [Specify which AWS Regions your account can use](#).

Você pode criar um cliente de serviço da AWS que vá para uma [região específica](#). Você também pode configurar seu aplicativo com uma região que será usada para [todos os clientes de serviço da AWS](#). Esses dois casos serão explicados a seguir.

Crie um cliente de serviço com uma região específica

Você pode especificar a região para qualquer um dos clientes de AWS serviço em seu aplicativo. Definir a Região desta maneira tem precedência sobre qualquer configuração global para aquele cliente de serviço particular.

Região existente

Este exemplo mostra como instanciar um cliente do [Amazon EC2](#) em uma região existente. Ele usa campos [RegionEndpoint](#) definidos.

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

Nova região usando a classe RegionEndpoint

Este exemplo mostra como estruturar um novo endpoint de região usando [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
}
```

Nova região usando a classe de configuração do cliente de serviço

Este exemplo mostra como usar a propriedade `ServiceURL` da classe de configuração do cliente de serviço para especificar a região; nesse caso, usando a classe [AmazonEC2Config](#).

Esta técnica funciona mesmo se o endpoint da região não seguir o padrão regular de endpoints da região.

```
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
}
```

Especifique uma região para todos os clientes de serviço

Há várias maneiras de especificar uma região para todos os clientes de serviço da AWS que seu aplicativo cria. Essa região é usada para clientes de serviço que não foram criados com uma região específica.

O AWS SDK para .NET procura um valor de região na seguinte ordem.

Perfis

Defina um perfil que seu aplicativo ou o SDK tenha carregado. Para obter mais informações, consulte [Resolução de perfil e credenciais](#).

Variáveis de ambiente

Defina na variável de ambiente `AWS_REGION`.

No Linux ou macOS:

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

No Windows:

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

Note

Se você definir essa variável de ambiente para todo o sistema (usando `export` ou `setx`), ela afetará todos os SDKs e toolkits, não apenas o AWS SDK para .NET.

Classe AWSConfigs

Defina como uma propriedade [AWSConfigs.AWSRegion](#).

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

Resolução da região

Se nenhum dos métodos descritos acima for usado para especificar uma Região da AWS, o AWS SDK para .NET tenta encontrar uma região na qual o cliente de serviço da AWS vai operar.

Ordem da resolução da região

1. Arquivos de configuração do aplicativo, como `app.config` e `web.config`.
2. Variáveis de ambiente (`AWS_REGION` e `AWS_DEFAULT_REGION`).
3. Um perfil com o nome especificado por um valor em `AWSConfigs.AWSProfileName`.
4. Um perfil com o nome especificado pela variável de ambiente `AWS_PROFILE`.
5. O perfil `[default]`.
6. Metadados da instância do Amazon EC2 (se estiver sendo executada em uma instância do EC2).

Se nenhuma região for encontrada, o SDK lançará uma exceção declarando que o cliente de serviço da AWS não tem uma região configurada.

Informações especiais sobre a região da China (Pequim)

Para usar serviços na região China (Pequim), é necessário ter uma conta e credenciais específicas para a região China (Pequim). As contas e credenciais para outras regiões da AWS não funcionarão para a região China (Pequim). Da mesma forma, as contas e credenciais para a região China (Pequim) não funcionarão em outras regiões da AWS. Para ter informações sobre os endpoints e protocolos que estão disponíveis na região China (Pequim), consulte [Endpoints da região Pequim](#).

Informações especiais sobre novos serviços da AWS

Novos serviços da AWS podem ser lançados inicialmente em algumas regiões e, em seguida, liberados para outras regiões. Nestes casos, não é necessário instalar o SDK mais recente para acessar as novas regiões para esse serviço. Você pode especificar regiões recém-adicionadas globalmente ou por cliente, como mostrado anteriormente.

Instale pacotes do AWSSDK com o NuGet

[O NuGet é um sistema de gerenciamento de pacotes para a plataforma .NET.](#) Com o NuGet, você pode instalar os [pacotes AWSSDK](#), bem como várias outras extensões, em seu projeto. Para obter informações adicionais, consulte o repositório [aws/dotnet](#) no site do GitHub.

O NuGet sempre tem as versões mais recentes dos pacotes AWSSDK, bem como as versões anteriores. O NuGet está ciente das dependências entre os pacotes e instala todos os pacotes necessários automaticamente.

Warning

A lista de pacotes NuGet pode incluir um chamado simplesmente “AWSSDK” (sem identificador anexado). NÃO instale esse pacote NuGet; ele é legado e não deve ser usado para novos projetos.

Os pacotes instalados com o NuGet são armazenados com seu projeto em vez de em um local central. Isso permite a instalação de versões do conjunto específicas para um certo aplicativo, sem

criar problemas de compatibilidade para outros aplicativos. Para obter mais informações sobre o NuGet, consulte a [documentação do NuGet](#).

 Note

Se não puder ou não tiver permissão para baixar e instalar pacotes NuGet por projeto, você poderá obter os assemblies do AWSSDK e armazená-los localmente (ou on-premises). Se isso se aplica a você e você ainda não tiver obtido os assemblies do AWSSDK, consulte [Como obter assemblies do AWSSDK](#). Para aprender a usar os assemblies armazenados localmente, consulte [Instale assemblies do AWSSDK sem o NuGet](#).

Como usar o NuGet pela linha de comando ou pelo terminal

1. Acesse os [pacotes AWSSDK no NuGet](#) e determine quais pacotes são necessários em seu projeto; por exemplo, [AWSSDK.S3](#).
2. Copie o comando da CLI do .NET da página desse pacote, como mostrado no exemplo a seguir.

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

3. No diretório do projeto, execute esse comando da CLI do .NET. O NuGet também instala quaisquer dependências, como o [AWSSDK.Core](#).

 Note

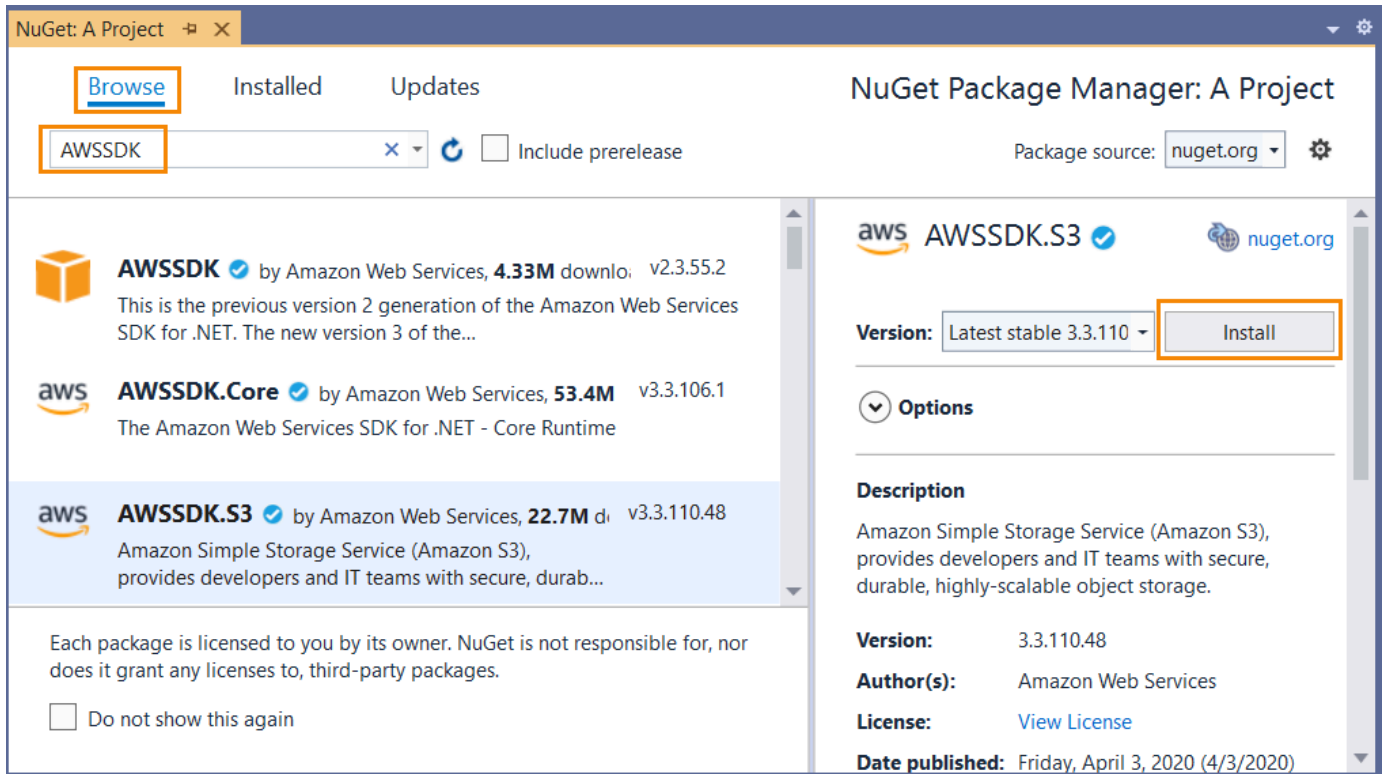
Se desejar somente a versão mais recente de um pacote NuGet, você poderá excluir as informações da versão do comando, conforme mostrado no exemplo a seguir.

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

Como usar o NuGet no Visual Studio Solution Explorer

1. No Solution Explorer, clique com o botão direito no projeto e, em seguida, selecione Manage NuGet Packages no menu de contexto.
2. No painel esquerdo do NuGet Package Manager (Gerenciador de pacotes NuGet), escolha Browse (Procurar). É possível usar a caixa de pesquisa para procurar o pacote que deseja instalar. O NuGet também instala quaisquer dependências, como o [AWSSDK.Core](#).

A figura a seguir mostra a instalação do pacote AWSSDK.S3.



Usar o NuGet a partir do Console de gerenciamento de pacotes

No Visual Studio, escolha Ferramentas, NuGet Package Manager, Package Manager Console.

Instale os pacotes do AWSSDK que você deseja a partir do Console de gerenciamento de pacotes usando o comando **Install-Package**. Por exemplo, para instalar o [AWSSDK.S3](#), use o comando a seguir.

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

O NuGet também instala quaisquer dependências, como o [AWSSDK.Core](#).

Se você precisar instalar uma versão anterior de um pacote, use a opção **-Version** e especifique a versão do pacote que você deseja, conforme mostrado no exemplo a seguir.

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

Para obter mais informações sobre comandos do Console do gerenciador de pacotes, consulte a [referência do PowerShell](#) na [documentação do NuGet](#) da Microsoft.

Instale assemblies do AWSSDK sem o NuGet

Este tópico descreve como você pode usar os assemblies do AWSSDK que você obteve e armazenou localmente (ou on-premises), conforme descrito em [Como obter assemblies do AWSSDK](#). Esse não é o método recomendado para lidar com referências do SDK, mas é obrigatório em alguns ambientes.

Note

O método recomendado para lidar com referências do SDK é baixar e instalar apenas os pacotes NuGet necessários para cada projeto. Esse método é descrito em [Instale pacotes do AWSSDK com o NuGet](#).

Para instalar os assemblies do AWSSDK

1. Crie uma pasta na área do projeto para os assemblies do AWSSDK necessários. Por exemplo, você pode chamar essa pasta de `AwsAssemblies`.
2. Se você ainda não tiver feito isso, [obtenha os assemblies do AWSSDK](#), que coloca os assemblies em alguma pasta local de download ou instalação. Copie os arquivos DLL dos assemblies necessários dessa pasta de download para o seu projeto (na pasta `AwsAssemblies`, em nosso exemplo).

Assegure-se de também copiar as dependências. Você pode encontrar informações sobre dependências no site do [GitHub](#).

3. Faça referência aos assemblies necessários da seguinte forma.

Cross-platform development

1. Abra o arquivo `.csproj` do seu projeto e adicione um elemento `<ItemGroup>`.
2. No elemento `<ItemGroup>`, adicione um elemento `<Reference>` com um atributo `Include` para cada assembly necessário.

Para o Amazon S3, por exemplo, você adicionaria as seguintes linhas ao arquivo `.csproj` do seu projeto.

No Linux e macOS:

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

No Windows:

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

3. Salve o arquivo .csproj do seu projeto.

Windows with Visual Studio and .NET Core

1. No Visual Studio, carregue seu projeto e abra Projeto, Adicionar referência.
2. Escolha o botão Procurar na parte inferior da caixa de diálogo. Navegue até a pasta do seu projeto e a subpasta para a qual você copiou os arquivos DLL necessários (AwsAssemblies, por exemplo).
3. Selecione todos os arquivos DLL, escolha Adicionar e escolha OK.
4. Salve o seu projeto.

Resolução de perfil e credenciais

O AWS SDK para .NET busca credenciais em uma determinada ordem e usa o primeiro conjunto disponível para o aplicativo atual.

Ordem de pesquisa de credencial

1. Credenciais definidas explicitamente no cliente do serviço da AWS, conforme descrito em [Acessar credenciais e perfis em um aplicativo](#).

Note

Esse tópico está na seção [Considerações especiais](#) porque não é o método preferencial para especificar credenciais.

2. Um perfil de credenciais com o nome especificado por um valor em [AWSConfigs.AWSProfileName](#).
3. Um perfil de credenciais com o nome especificado pela variável de ambiente `AWS_PROFILE`.
4. O perfil de credenciais `[default]`.
5. [SessionAWSCredentials](#) criadas a partir das variáveis de ambiente `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`, se todas forem não vazias.
6. [BasicAWSCredentials](#) criadas a partir das variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY`, se ambas forem não vazias.
7. O [provedor de credenciais de contêiner](#).
8. Metadados da instância do Amazon EC2.

Se o aplicativo estiver em execução em uma instância do Amazon EC2, como em um ambiente de produção, use um perfil do IAM conforme descrito em [Conceder acesso utilizando um perfil do IAM](#). Caso contrário, como em testes de pré-lançamento, armazene suas credenciais em um arquivo que use o formato de arquivo de credenciais da AWS ao qual seu aplicativo web tenha acesso no servidor.

Resolução do perfil

Com dois mecanismos de armazenamento diferentes para credenciais, é importante entender como configurar o AWS SDK para .NET para usá-los. A propriedade [AWSConfigs.AWSProfilesLocation](#) controla como o AWS SDK para .NET encontra perfis de credenciais.

AWSProfilesLocation	Comportamento da resolução do perfil
nulo (não definido) ou vazio	Pesquise na SDK Store se a plataforma for compatível e, em seguida, pesquise o arquivo de credenciais compartilhado da AWS no local padrão . Se o perfil não estiver em nenhum desses locais, busque <code>~/.aws/config</code>

AWSPProfilesLocation	Comportamento da resolução do perfil
	(Linux ou macOS) ou %USERPROFILE%\ .aws \config (Windows).
O caminho para um arquivo no formato de arquivo de credenciais da AWS	Pesquise somente o arquivo especificado para um perfil com o nome especificado.

Usar credenciais da conta de usuário federado

Os aplicativos que usam o AWS SDK para .NET ([AWSSDK.Core](#) versão 3.1.6.0 ou posterior) podem usar contas de usuário federado por meio do Active Directory Federation Services (AD FS - Serviços de federação do Active Directory) para acessar web services da AWS usando o Security Assertion Markup Language (SAML).

Suporte ao acesso federado significa que usuários podem se autenticar usando o Active Directory. São concedidas credenciais temporárias ao usuário automaticamente. Estas credenciais temporárias, que são válidas por uma hora, são usadas quando o aplicativo invoca os serviços da AWS. O SDK faz o gerenciamento das credenciais temporárias. Para contas de usuário associados a um domínio, se o aplicativo realizar uma chamada mas as credenciais expiraram, o usuário é reautenticado automaticamente e são concedidas novas credenciais. (Para contas que não estão associadas a um domínio, o usuário deve informar as credenciais antes da reautenticação.)

Para usar esse suporte no aplicativo .NET, primeiro é necessário configurar o perfil da função usando um cmdlet PowerShell. Para saber como, consulte a [documentação do AWS Tools for Windows PowerShell](#).

Depois de configurar o perfil da função, faça referência ao perfil em seu aplicativo. Há várias maneiras de fazer isso, uma delas é usar a propriedade [AWSConfigs.AWSProfileName](#) da mesma forma que você faria com outros perfis de credenciais.

A assembly AWS Security Token Service ([AWSSDK.SecurityToken](#)) fornece o suporte SAML para obter credenciais da AWS. Para usar as credenciais da conta de usuário federado, certifique-se de que esse assembly esteja disponível para seu aplicativo.

Especificar funções ou credenciais temporárias

Para aplicativos executados em instâncias do Amazon EC2, a forma mais segura de gerenciar as credenciais é utilizar perfis do IAM, conforme descrito em [Conceder acesso utilizando um perfil do IAM](#).

Para cenários do aplicativo em que o executável do software está disponível para usuários de fora da organização, recomendamos que você projete o software para usar credenciais de segurança temporárias. Além de oferecer acesso restrito aos recursos da AWS, essas credenciais têm o benefício de expirarem após um período especificado. Para obter mais informações sobre as credenciais de segurança temporárias, consulte:

- [Credenciais de segurança temporárias](#)
- [Banco de identidades do Amazon Cognito](#)

Usar credenciais de proxy

Se o seu software se comunica com a AWS por meio de um proxy, especifique as credenciais para o proxy usando a propriedade `ProxyCredentials` na classe `Config` para um serviço. A classe `Config` de um serviço normalmente faz parte do namespace primário do serviço. Os exemplos incluem o seguinte: [AmazonCloudDirectoryConfig](#) no namespace [Amazon.CloudDirectory](#) e [AmazonGameLiftConfig](#) no namespace [Amazon.GameLift](#).

Para [Amazon S3](#), por exemplo, você pode usar código semelhante ao seguinte, quando `SecurelyStoredUserName` e `SecurelyStoredPassword` são o nome de usuário e senha do proxy especificados em um objeto [NetworkCredential](#).

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

Note

Versões anteriores do SDK utilizavam `ProxyUsername` e `ProxyPassword`, mas estas propriedades estão obsoletas.

Informações adicionais sobre usuários e perfis

Para fazer o desenvolvimento do .NET na AWS ou para executar aplicativos .NET na AWS, você precisa ter uma combinação de usuários, conjuntos de permissões e perfis de serviço que sejam apropriados para essas tarefas.

Os usuários específicos, os conjuntos de permissões e os perfis de serviço que você cria e a maneira como você os utiliza dependerão dos seus aplicativos. Veja a seguir algumas informações adicionais sobre por que eles podem ser usados e como criá-los.

Usuários e conjuntos de permissões

Embora seja possível usar uma conta de usuário do IAM com credenciais de longo prazo para acessar os serviços da AWS, essa não é mais uma prática recomendada e deve ser evitada. Mesmo durante o desenvolvimento, é prática recomendada criar usuários e conjuntos de permissões no Centro de Identidade do AWS IAM e usar credenciais temporárias fornecidas por uma fonte de identidade.

Para desenvolvimento, você pode usar o usuário que criou ou recebeu em [Configure a autenticação do SDK](#). Se você tiver permissões apropriadas do Console de gerenciamento da AWS, também poderá criar conjuntos de permissões diferentes com privilégio mínimo para esse usuário ou criar usuários especificamente para projetos de desenvolvimento, fornecendo conjuntos de permissões com privilégio mínimo. A ação que você escolher, se for o caso, dependerá das circunstâncias.

Para obter mais informações sobre esses usuários e conjuntos de permissões e como criá-los, consulte [Autenticação e acesso](#) no Guia de referência de AWS SDKs e ferramentas e [Conceitos básicos](#) no Guia do usuário do Centro de Identidade do AWS IAM.

Perfis de serviço

Você pode configurar um perfil de serviço da AWS para acessar serviços da AWS em nome dos usuários. Esse tipo de acesso é apropriado quando várias pessoas estão executando a aplicação remotamente; por exemplo, em uma instância do Amazon EC2 que você criou para essa finalidade.

O processo de criação de um perfil de serviço varia de acordo com a situação, mas é basicamente o seguinte.

1. Faça login no Console de gerenciamento da AWS e abra o console do IAM, em <https://console.aws.amazon.com/iam/>.

2. Selecione Funções e, depois, Criar função.
3. Escolha Serviço da AWS, encontre e selecione EC2 (por exemplo) e, depois, selecione o caso de uso do EC2 (por exemplo).
4. Escolha Próximo: Permissões e selecione as [políticas apropriadas](#) para os serviços da AWS que seu aplicativo usará.

 Warning

NÃO escolha a política AdministratorAccess porque ela concede permissões de leitura e gravação a quase tudo na sua conta.

5. Escolha Próximo: Tags e insira as tags que você desejar.

Você encontrará informações sobre tags em [Controlar o acesso a recursos da AWS usando tags](#) no [Manual do usuário do IAM](#).

6. Escolha Próximo: Revisar para fornecer um nome de perfil e uma descrição de perfil. Então, escolha Criar perfil.

Você pode encontrar informações de alto nível sobre os perfis do IAM em [Identities \(usuários, grupos e perfis\)](#) no [Guia do usuário do IAM](#). Encontre informações detalhadas sobre perfis nesse tópico [Perfis do IAM](#).

Informações adicionais sobre perfis

- Use [perfis do IAM para tarefas](#) do Amazon Elastic Container Service (Amazon ECS).
- Use [perfis do IAM](#) para aplicações em execução nas instâncias do Amazon EC2.

Configurações avançadas para seu projeto do AWS SDK para .NET

Os tópicos desta seção contêm informações sobre tarefas e métodos de configurações adicionais que podem ser do seu interesse.

Tópicos

- [Usando AWSSDK.Extensions.NETCore.Configuração e interface IConfiguration](#)
- [Configuração de outros parâmetros do aplicativo](#)

- [Referência dos arquivos de configuração para o AWS SDK para .NET](#)

Usando AWSSDK.Extensions.NETCore.Configuração e interface IConfiguration

(Este tópico era anteriormente intitulado “Configurando o AWS SDK para .NET com o.NET Core”)

Uma das principais alterações no .NET Core é a retirada do ConfigurationManager e dos arquivos padrão app.config e web.config usados nos aplicativos .NET Framework e ASP.NET.

A configuração no .NET Core é baseada em pares de chave/valor estabelecidos pelos provedores de configuração. Os provedores de configuração leem dados de configuração em pares de chave/valor de uma variedade de fontes de configuração, incluindo argumentos de linha de comando, arquivos de diretório, variáveis de ambiente e arquivos de configurações.

Note

Para obter mais informações, consulte [Configuração no ASP.NET Core](#).

Para facilitar o uso do AWS SDK para .NET com o.NET Core, você pode usar as [AWSSDK.Extensions.NETCore NuGet .Pacote de configuração](#). Como muitas bibliotecas.NET Core, ela adiciona métodos de extensão à IConfiguration interface para facilitar a obtenção da AWS configuração.

O código-fonte desse pacote está GitHub em<https://github.com/aws/aws-sdk-net/tree/aws-sdk-net-v3.7/extensions/src/AWSSDK.Extensions.NETCore.Setup>.

Usando AWSSDK.Extensions.NETCore.Configuração

Suponha que você crie um aplicativo ASP.NET Core Model-View-Controller (MVC), o que pode ser feito com o modelo de aplicativo Web ASP.NET Core no Visual Studio ou executando na dotnet new mvc ... CLI do.NET Core. Quando você cria tal aplicativo, o construtor do Startup.cs lida com a configuração lendo várias fontes de entrada de provedores de configuração, como o appsettings.json.

```
public Startup(IConfiguration configuration)
{
```

```
Configuration = configuration;  
}
```

Para usar o `Configuration` objeto para obter as `AWS` opções, primeiro adicione o `AWSSDK.Extensions.NETCore.Setup` NuGet pacote. Em seguida, adicione as opções ao arquivo de configuração, conforme descrito a seguir.

Observe que um dos arquivos adicionados ao seu projeto é `appsettings.Development.json`. Isso corresponde a um `EnvironmentName` definido como `Desenvolvimento`. Durante o desenvolvimento, você coloca sua configuração nesse arquivo, que só é lido durante o teste local. Quando você implanta uma instância do Amazon EC2 que foi `EnvironmentName` configurada como `Produção`, esse arquivo é ignorado e volta para as `AWS` SDK para .NET credenciais e a região do IAM que estão configuradas para a instância do Amazon EC2.

As definições de configuração a seguir mostram exemplos dos valores que você pode incluir no arquivo `appsettings.Development.json` em seu projeto para fornecer as configurações da `AWS`.

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

Para acessar uma configuração em um arquivo `CSHTML`, use a diretiva `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>
```

Para acessar as `AWS` opções definidas no arquivo a partir do código, chame o método de `GetAWSOptions` extensão adicionado `IConfiguration` a.

Para construir um cliente de serviço a partir dessas opções, chame `CreateServiceClient`. O código de exemplo a seguir mostra como criar um cliente de serviço do Amazon S3. (Certifique-se de adicionar o [AWSSDK NuGet pacote.S3](#) ao seu projeto.)

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

Também é possível criar vários clientes de serviço com configurações incompatíveis ao usar várias entradas no arquivo `appsettings.Development.json`, conforme mostrado nos exemplos a seguir, em que a configuração de `service1` inclui a região `us-west-2` e a configuração de `service2` inclui o URL do endpoint especial.

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

Depois, você pode obter as opções para um serviço específico usando a entrada no arquivo JSON. Por exemplo, para obter as configurações para `service1`, use o seguinte.

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

Valores permitidos no arquivo appsettings

Os valores de configuração do aplicativo a seguir podem ser definidos no arquivo `appsettings.Development.json`. Os nomes de campo devem usar a capitalização mostrada. Para obter mais detalhes sobre essas configurações, consulte a classe [AWS.Runtime.ClientConfig](#).

- Região
- Perfil
- ProfilesLocation

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

Injeção de dependência no núcleo do ASP.NET

As AWSSDKextensões. NETCoreO NuGet pacote.Setup também se integra a um novo sistema de injeção de dependência no ASP.NET Core. O método ConfigureServices na classe Startup do seu aplicativo é o local onde os serviços MVC são adicionados. Se o aplicativo usa a Estrutura de entidade, também é aqui que ela é inicializada.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
}
```

Note

O histórico sobre a injeção de dependência no .NET Core está disponível no [site de documentação do .NET Core](#).

O `AWSSDK.Extensions.NETCore.Setup` NuGet pacote adiciona novos métodos de extensão `IServiceCollection` que você pode usar para adicionar AWS serviços à injeção de dependência. O código a seguir mostra como adicionar as AWS opções que são lidas `IConfiguration` para adicionar o Amazon S3 e o DynamoDB à lista de serviços. (Certifique-se de adicionar os DBv2 NuGet pacotes [AWSSDK.S3](#) e [AWSSDK.Dynamo](#) ao seu projeto.)

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    services.AddDefaultAWSOptions(Configuration.GetAWSOptions());
    services.AddAWSService<IAmazonS3>();
    services.AddAWSService<IAmazonDynamoDB>();
}
```

Agora, se os controladores MVC usam `IAmazonS3` ou `IAmazonDynamoDB` como parâmetros em seus construtores, o sistema de injeção de dependência passa nesses serviços.

```
public class HomeController : Controller
{
    IAmazonS3 S3Client { get; set; }

    public HomeController(IAmazonS3 s3Client)
    {
        this.S3Client = s3Client;
    }

    ...
}
```

Configuração de outros parâmetros do aplicativo

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework. Os arquivos `App.config` e `Web.config` não estão presentes por padrão em projetos baseados no .NET Core.

Abra para ver o conteúdo do .NET Framework

Há diversos parâmetros do aplicativo que você pode configurar:

- [AWSLogging](#)
- [AWSLogMetrics](#)
- [AWSRegion](#)
- [AWSResponseLogging](#)
- [AWS.DynamoDBContext.TableNamePrefix](#)
- [AWS.S3.UseSignatureVersion4](#)
- [AWSEndpointDefinition](#)
- [AWS Endpoints gerados por serviços](#)

Esses parâmetros podem ser configurados no arquivo `App.config` ou `Web.config` do aplicativo. Embora você também possa configurá-los com a AWS SDK para .NET API, recomendamos que você use o `.config` arquivo do aplicativo. Ambas as abordagens são descritas aqui.

Para obter mais informações sobre o uso do elemento `<aws>` conforme descrito posteriormente neste tópico, consulte [Referência de arquivos de configuração para o AWS SDK para .NET](#).

AWSLogging

Configura a forma como o SDK registra os eventos, caso registre. Por exemplo, a abordagem recomendada é usar o elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

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

Alternativa:

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

Os valores possíveis são:

None

Desative o registro de eventos do . Esse é o padrão.

log4net

Registre usando o log4net.

SystemDiagnostics

Registre usando a classe `System.Diagnostics`.

Defina diversos valores para o atributo `logTo`, separado por vírgulas. O exemplo a seguir define ambos os registros `log4net` e `System.Diagnostics` no arquivo `.config`:

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

Alternativa:

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

[Como alternativa, usando a AWS SDK para .NET API, combine os valores da `LoggingOptions` enumeração e defina a `AWSConfigs` propriedade `.Logging`:](#)

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

As alterações nessa configuração entram em vigor somente para novas instâncias AWS do cliente.

AWSLogMétricas

Especifica se o SDK deve ou não registrar as métricas de desempenho. Para definir a configuração do registro das métricas no arquivo `.config`, defina o valor do atributo `logMetrics` no elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

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

Como alternativa, defina a chave `AWSLogMetrics` na seção `<appSettings>`:

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

Como alternativa, para definir o registro de métricas com a AWS SDK para .NET API, defina [AWSConfigso. LogMetrics](#) propriedade:


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

Essa definição configura propriedade padrão `LogMetrics` para todos os clientes/configurações. As alterações nessa configuração entram em vigor somente para novas instâncias AWS do cliente.

AWSRegion

Configura a AWS região padrão para clientes que não especificaram explicitamente uma região. Para definir a região no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `region` no elemento `aws`:

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

Como alternativa, defina a chave `AWSRegion` na seção `<appSettings>`:

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

Como alternativa, para definir a região com a AWS SDK para .NET API, defina [AWSConfigso.AWSRegion](#) propriedade:

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

Para obter mais informações sobre como criar um AWS cliente para uma região específica, consulte [Seleção de AWS região](#). As alterações nessa configuração entram em vigor somente para novas instâncias AWS do cliente.

AWSResponseRegistro

Configura quando o SDK deve registrar respostas de serviços. Os valores possíveis são:

Never

Nunca registrar as respostas de serviços. Esse é o padrão.

Always

Sempre registrar as respostas de serviços.

OnError

Somente registrar as respostas de serviços quando ocorrer um erro.

Para definir a configuração do registro de serviços no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `logResponses` no elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

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

Como alternativa, defina a chave de `AWSResponseLogging` na `<appSettings>` seção:

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

Como alternativa, para definir o registro de serviços com a AWS SDK para .NET API, defina [AWSConfigs.ResponseLogging](#) propriedade de um dos valores da [ResponseLoggingOption](#) enumeração:

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

As alterações desta configuração entram em vigor imediatamente.

AWS.DynamoDBContext.TableNamePrefix

Configura a `TableNamePrefix` padrão que o `DynamoDBContext` usará, se não for configurada manualmente.

Para definir o prefixo do nome da tabela no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `tableNamePrefix` no elemento `<dynamoDBContext>`, que é um elemento filho do elemento `<dynamoDB>`, que por sua vez é um elemento filho do elemento `<aws>`:

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

Como alternativa, defina a chave `AWS.DynamoDBContext.TableNamePrefix` na seção `<appSettings>`:

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

Como alternativa, para definir o prefixo do nome da tabela com a AWS SDK para .NET API, defina a propriedade [AWSConfigsDBContextTableNamePrefix.Dynamo](#):

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

As alterações desta configuração serão implementadas somente em instâncias recentemente criadas do `DynamoDBContextConfig` e `DynamoDBContext`.

AWS.S3.UseSignatureVersion4

Configura se o cliente do Amazon S3 deve ou não usar a assinatura do Signature versão 4 com as solicitações.

Para definir a assinatura da versão 4 para o Amazon S3 no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `useSignatureVersion4` do elemento `<s3>`, que é um elemento filho do elemento `<aws>`:

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

Como alternativa, defina a chave `AWS.S3.UseSignatureVersion4` como `true` na seção `<appSettings>`:

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

Como alternativa, para definir a assinatura da versão 4 com a AWS SDK para .NET API, defina a propriedade [AWSConfigs.S3 UseSignatureVersion4](#) como: `true`

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

Por padrão, esta configuração é `false`, mas o Signature versão 4 pode ser usado como padrão em alguns casos ou em algumas regiões. Quando a configuração é `true`, o Signature versão 4 será usada em todas as solicitações. As alterações desta configuração entram em vigor somente para novas instâncias de cliente do Amazon S3.

AWSEndpointDefinição

Configura se o SDK deve ou não usar um arquivo de configuração personalizado que define as regiões e os endpoints.

Para definir o arquivo de definição do endpoint no arquivo `.config`, recomendamos definir o valor do atributo `endpointDefinition` no elemento `<aws>`.

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

Como alternativa, você pode definir a chave de AWSEndpointdefinição na <appSettings> seção:

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

Como alternativa, para definir o arquivo de definição do endpoint com a AWS SDK para .NET API, defina o. AWSConfigs EndpointDefinitionpropriedade:

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

Se nenhum nome de arquivo for fornecido, não será usado um arquivo de configuração personalizado. As alterações nessa configuração entram em vigor somente para novas instâncias AWS do cliente. O arquivo endpoint.json está disponível em <https://github.com/aws/aws-sdk-net/blob/aws-sdk-net-v3.7/sdk/src/Core/endpoints.json>.

AWS Endpoints gerados por serviços

Alguns AWS serviços geram seus próprios endpoints em vez de consumir um endpoint regional. Os clientes desses serviços consomem um URL de serviço específico ao serviço e aos recursos. Dois exemplos desses serviços são Amazon CloudSearch AWS IoT e. Os exemplos a seguir mostram como obter os endpoints para esses serviços.

Exemplo de CloudSearch endpoints da Amazon

O CloudSearch cliente da Amazon é usado para acessar o serviço de CloudSearch configuração da Amazon. Você usa o serviço CloudSearch de configuração da Amazon para criar, configurar e gerenciar domínios de pesquisa. Para criar um domínio de pesquisa, crie um [CreateDomainRequest](#)objeto e forneça a DomainName propriedade. Crie um [AmazonCloudSearchClient](#)objeto usando o objeto de solicitação. Chame o método [CreateDomain](#). O [CreateDomainResponse](#)objeto retornado da chamada contém uma DomainStatus propriedade que tem os SearchService pontos finais DocService e. Crie um [AmazonCloudSearchDomainConfig](#)objeto e use-o para inicializar DocService as SearchService instâncias da [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 Exemplo de endpoints

Para obter o endpoint para AWS IoT, crie um [AmazonIoTClient](#) objeto e chame o [DescribeEndPoint](#) método. O [DescribeEndPointResponse](#) objeto retornado contém `EndpointAddress` o. Crie um [AmazonIotDataConfig](#) objeto, defina a `ServiceURL` propriedade e use o objeto para instanciar a [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");
}
```

Referência dos arquivos de configuração para o AWS SDK para .NET

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework. Os arquivos `App.config` e `Web.config` não estão presentes por padrão em projetos baseados no .NET Core.

Abra para ver o conteúdo do .NET Framework

Use um arquivo `App.config` ou `Web.config` de projeto .NET para especificar as configurações da AWS, como credenciais da AWS, opções de registro em log, endpoints de serviço da AWS e regiões da AWS, bem como algumas configurações dos serviços da AWS, como Amazon DynamoDB, Amazon EC2 e Amazon S3. As informações a seguir descrevem como formatar de forma apropriada um arquivo `App.config` ou `Web.config` para especificar esses tipos de configurações.

Note

Embora seja possível continuar a usar o elemento `<appSettings>` em um arquivo `App.config` ou `Web.config` para especificar as configurações da AWS, recomendamos o uso dos elementos `<configSections>` e `<aws>` conforme descrito posteriormente neste tópico. Para obter mais informações sobre o elemento `<appSettings>`, consulte os exemplos do elemento `<appSettings>` em [Configuração do seu aplicativo AWS SDK para .NET](#).

Note

Embora seja possível continuar a usar as propriedades a seguir da classe [AWSConfigs](#) em um arquivo de código para especificar as configurações da AWS, estas propriedades são obsoletas e podem não comportar versões futuras:

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

Geralmente, recomendamos que em vez de usar as propriedades da classe `AWSConfigs` em um arquivo de código para especificar as configurações da AWS, use os elementos `<configSections>` e `<aws>` em um arquivo `App.config` ou `Web.config` para especificar as configurações da AWS, conforme descrito posteriormente neste tópico. Para

obter mais informações sobre as propriedades anteriores, consulte os exemplos de códigos da AWSConfigs em [Configuração do seu aplicativo AWS SDK para .NET](#).

Tópicos

- [Declarar uma seção de configurações da AWS](#)
- [Elementos permitidos](#)
- [Referência dos elementos](#)

Declarar uma seção de configurações da AWS

Especifique as configurações da AWS em um arquivo `App.config` ou `Web.config` a partir do elemento `<aws>`. Antes de começar a usar o elemento `<aws>`, é necessário criar um elemento `<section>` (que é um elemento filho do elemento `<configSections>`) e definir o seu atributo `name` como `aws` e o atributo `type` como `Amazon.AWSSection, AWSSDK.Core`, conforme mostrado no seguinte exemplo:

```
<?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>
```

O editor do Visual Studio não oferece a conclusão de código automática (IntelliSense) para o elemento `<aws>` ou para os seus elementos filho.

Para auxiliá-lo na criação de uma versão corretamente formatada `<aws>`, chame o método `Amazon.AWSConfigs.GenerateConfigTemplate`. Isso gera como saída uma versão canônica do elemento `<aws>` como uma string formatada, que pode ser adaptada para as suas necessidades. As seções a seguir descrevem os atributos e os elementos filho do elemento `<aws>`.

Elementos permitidos

A seguir encontra-se uma lista das relações lógicas entre os elementos permitidos em uma seção de configurações da AWS. Gere a versão mais recente dessa lista chamando o método `Amazon.AWSConfigs.GenerateConfigTemplate`, que gera como saída uma versão canônica do elemento `<aws>` como uma string que pode ser adaptada para as suas necessidades.

```
<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>
```

Referência dos elementos

A seguir encontra-se uma lista dos elementos permitidos em uma seção de configurações da AWS. Para cada elemento, são listados os atributos permitidos e os elementos filho.

Tópicos

- [alias](#)
- [aws](#)
- [dynamoDB](#)
- [dynamoDBContext](#)
- [ec2](#)
- [registro em log](#)
- [mapear](#)
- [property](#)
- [proxy](#)
- [s3](#)

alias

O elemento `<alias>` representa um único item em um conjunto de um ou mais mapeamentos from-table até to-table, que especifica uma tabela diferente daquela configurada para um tipo. Esse elemento é mapeado para uma instância da classe `Amazon.Util.TableAlias` da propriedade `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` no AWS SDK para .NET. O remapeamento é feito antes de aplicar um prefixo de nome da tabela.

Esse elemento pode incluir os seguintes atributos:

fromTable

A parte from-table do mapeamento from-table até to-table. Esse atributo é mapeado para a propriedade `Amazon.Util.TableAlias.FromTable` no AWS SDK para .NET.

toTable

A parte to-table do mapeamento from-table até to-table. Esse atributo é mapeado para a propriedade `Amazon.Util.TableAlias.ToTable` no AWS SDK para .NET.

O pai do elemento `<alias>` é o elemento `<tableAliases>`.

O elemento `<alias>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<alias>` em uso:

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

aws

O elemento `<aws>` representa o elemento mais superior em uma seção de configurações da AWS. Esse elemento pode incluir os seguintes atributos:

endpointDefinition

O caminho absoluto para um arquivo de configuração personalizado que define as regiões e endpoints da AWS para o uso. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.EndpointDefinition` no AWS SDK para .NET.

profileName

O nome do perfil para as credenciais da AWS armazenadas que serão utilizadas para realizar chamadas de serviços. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSProfileName` no AWS SDK para .NET.

profilesLocation

O caminho absoluto do local do arquivo de credenciais compartilhado com outros SDKs da AWS. Por padrão, o arquivo de credenciais está armazenado no diretório `.aws` no diretório base do usuário atual. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSProfilesLocation` no AWS SDK para .NET.

region

O ID da região da AWS padrão para clientes que não especificaram explicitamente uma região. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSRegion` no AWS SDK para .NET.

O elemento `<aws>` não possui elemento pai.

O elemento `<aws>` pode incluir os seguintes elementos filho:

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

Veja a seguir um exemplo do elemento `<aws>` em uso:

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

dynamoDB

O elemento `<dynamoDB>` representa um conjunto de configurações do Amazon DynamoDB. Este elemento pode incluir o atributo `conversionSchema`, que representa a versão a ser usada para conversão entre objetos .NET e DynamoDB. Os valores permitidos incluem V1 e V2. Esse atributo é mapeado para a classe `Amazon.DynamoDBv2.DynamoDBEntryConversion` no AWS SDK para .NET. Para obter mais informações, consulte [Série DynamoDB – esquemas de conversão](#).

O pai do elemento `<dynamoDB>` é o elemento `<aws>`.

O elemento `<dynamoDB>` pode incluir o elemento filho `<dynamoDBContext>`.

Veja a seguir um exemplo do elemento `<dynamoDB>` em uso:

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

dynamoDBContext

O elemento `<dynamoDBContext>` representa um conjunto de configurações específicas de contexto do Amazon DynamoDB. Este elemento pode incluir o atributo `tableNamePrefix`, que representa o prefixo do nome da tabela padrão que o contexto do DynamoDB usará se não for configurado manualmente. Esse atributo é mapeado para a propriedade `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` da propriedade `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` no AWS SDK para .NET. Para obter mais informações, consulte [Aprimoramentos do DynamoDB SDK](#).

O pai do elemento `<dynamoDBContext>` é o elemento `<dynamoDB>`.

O elemento `<dynamoDBContext>` pode incluir os seguintes elementos filho:

- `<alias>` (uma ou mais instâncias)
- `<map>` (uma ou mais instâncias)

Veja a seguir um exemplo do elemento `<dynamoDBContext>` em uso:

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

ec2

O elemento `<ec2>` representa um conjunto de configurações do Amazon EC2. Este elemento pode incluir o atributo `useSignatureVersion4`, que especifica se a assinatura do Signature Version4 será usada para todas as solicitações (verdadeiro) ou se ela não será usada para todas as solicitações (falso, o padrão). Esse atributo é mapeado para a propriedade `Amazon.Util.EC2Config.UseSignatureVersion4` da propriedade `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` no AWS SDK para .NET.

O pai do elemento `<ec2>` é o elemento.

O elemento `<ec2>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<ec2>` em uso:

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

registro em log

O elemento `<logging>` representa um conjunto de configurações para o registro de respostas e de métricas de desempenho. Esse elemento pode incluir os seguintes atributos:

logMetrics

Se as métricas de desempenho serão registradas para todos os clientes e configurações (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetrics` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogMetrics` no AWS SDK para .NET.

logMetricsCustomFormatter

O tipo de dados e o nome do conjunto de um formatador personalizado para as métricas de registro. Esse atributo é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` no AWS SDK para .NET.

logMetricsFormat

O formato em que as métricas de registro são apresentadas (é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetricsFormat` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` no AWS SDK para .NET).

Os valores permitidos incluem:

JSON

Use o formato JSON.

Standard

Use o formato padrão.

logResponses

Quando registrar as respostas de serviços (é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogResponses` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogResponses` no AWS SDK para .NET).

Os valores permitidos incluem:

Always

Sempre registrar as respostas de serviços.

Never

Nunca registrar as respostas de serviços.

OnError

Somente registrar as respostas de serviços quando existirem erros.

logTo

Onde registrar (é mapeado para a propriedade `LogTo` a partir da propriedade `Amazon.AWSConfigs.LoggingConfig.LogTo` no AWS SDK para .NET).

Os valores permitidos incluem um ou mais entre:

Log4Net

Registrar no log4net.

None

Desativar o registro.

SystemDiagnostics

Registrar no `System.Diagnostics`.

O pai do elemento `<logging>` é o elemento `<aws>`.

O elemento `<logging>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<logging>` em uso:

```
<logging  
  logTo="SystemDiagnostics"
```

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

mapear

O elemento `<map>` representa um único item em um conjunto de mapeamentos type-to-table de tipos .NET para tabelas do DynamoDB (é mapeado para uma instância da classe `TypeMapping` da propriedade `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` no AWS SDK para .NET). Para obter mais informações, consulte [Aprimoramentos do DynamoDB SDK](#).

Esse elemento pode incluir os seguintes atributos:

targetTable

A tabela do DynamoDB a qual o mapeamento se aplica. Esse atributo é mapeado para a propriedade `Amazon.Util.TypeMapping.TargetTable` no AWS SDK para .NET.

type

O tipo e o nome do conjunto ao qual o mapeamento se aplica. Esse atributo é mapeado para a propriedade `Amazon.Util.TypeMapping.Type` no AWS SDK para .NET.

O pai do elemento `<map>` é o elemento `<dynamoDBContext>`.

O elemento `<map>` pode incluir uma ou mais instâncias do elemento filho `<property>`.

Veja a seguir um exemplo do elemento `<map>` em uso:

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

property

O elemento `<property>` representa uma propriedade do DynamoDB. (Este elemento é mapeado para uma instância da classe `Amazon.Util.PropertyConfig` do método `AddProperty` no AWS SDK para .NET) Para obter mais informações, consulte [Aprimoramentos do DynamoDB SDK](#) e [Atributos do DynamoDB](#).

Esse elemento pode incluir os seguintes atributos:

attribute

O nome de um atributo da propriedade, como o nome de uma chave de intervalo. Esse atributo é mapeado para a propriedade `Amazon.Util.PropertyConfig.Attribute` no AWS SDK para .NET.

converter

O tipo de conversor que deve ser usado para esta propriedade. Esse atributo é mapeado para a propriedade `Amazon.Util.PropertyConfig.Converter` no AWS SDK para .NET.

ignore

Se a propriedade associada deve ser ignorada (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade `Amazon.Util.PropertyConfig.Ignore` no AWS SDK para .NET.

name

O nome da propriedade. Esse atributo é mapeado para a propriedade `Amazon.Util.PropertyConfig.Name` no AWS SDK para .NET.

version

Se essa propriedade deve armazenar o número da versão do item (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade `Amazon.Util.PropertyConfig.Version` no AWS SDK para .NET.

O pai do elemento `<property>` é o elemento `<map>`.

O elemento `<property>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<property>` em uso:

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

proxy

O elemento `<proxy>` representa as configurações necessárias de um proxy para o seu uso pelo AWS SDK para .NET. Esse elemento pode incluir os seguintes atributos:

host

O nome do host ou endereço IP do servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Host` da propriedade `Amazon.AWSConfigs.ProxyConfig.Host` no AWS SDK para .NET.

password

A senha para a autenticação com o servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Password` da propriedade `Amazon.AWSConfigs.ProxyConfig.Password` no AWS SDK para .NET.

porta

O número da porta do proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Port` da propriedade `Amazon.AWSConfigs.ProxyConfig.Port` no AWS SDK para .NET.

username

O nome de usuário para a autenticação com o servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Username` da propriedade `Amazon.AWSConfigs.ProxyConfig.Username` no AWS SDK para .NET.

O pai do elemento `<proxy>` é o elemento `<aws>`.

O elemento `<proxy>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<proxy>` em uso:

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

s3

O elemento `<s3>` representa um conjunto de configurações do Amazon S3. Este elemento pode incluir o atributo `useSignatureVersion4`, que especifica se a assinatura do Signature Version4 será usada para todas as solicitações (verdadeiro) ou se ela não será usada para todas as solicitações (falso, o padrão). Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` no AWS SDK para .NET.

O pai do elemento `<s3>` é o elemento `<aws>`.

O elemento `<s3>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<s3>` em uso:

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

Uso de credenciais herdadas

Os tópicos desta seção fornecem informações sobre como usar credenciais de longo ou curto prazo sem usar o Centro de Identidade do AWS IAM.

Warning

Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como [Centro de Identidade do AWS IAM](#).

Note

As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais sobre credenciais de curto e longo prazo, consulte [Outras formas de autenticação no Guia](#) de referência de ferramentas AWS SDKs e ferramentas.

Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito em [Configure a autenticação do SDK](#).

Avisos e orientações importantes para credenciais

Avisos para credenciais

- NÃO use as credenciais raiz da sua conta para acessar AWS recursos. Estas credenciais fornecem acesso ilimitado à conta e são difíceis de revogar.

- NÃO coloque chaves de acesso literais ou informações de credenciais em arquivos de aplicações. Se colocar, criará um risco de exposição acidental das credenciais se, por exemplo, fizer upload do projeto em um repositório público.
- NÃO inclua arquivos que contenham credenciais em sua área de projeto.
- Esteja ciente de que todas as credenciais armazenadas no `AWS credentials` arquivo compartilhado são armazenadas em texto simples.

Orientação adicional para gerenciar credenciais com segurança

[Para uma discussão geral sobre como gerenciar AWS credenciais com segurança, consulte Credenciais de AWS segurança em e Melhores práticas de segurança Referência geral da AWS](#)
[Se casos de uso no Guia do usuário do IAM.](#) Além dessas discussões, considere o seguinte:

- Crie usuários adicionais, como usuários no Centro de Identidade do IAM, e use as credenciais deles em vez de usar suas credenciais de usuário raiz da AWS . As credenciais de outros usuários poderão ser revogadas, se necessário, ou são temporárias por natureza. Além disso, você pode aplicar uma política a cada usuário para acessar somente determinados recursos e ações e, assim, adotar uma postura de permissões com privilégio mínimo.
- Use [perfis do IAM para tarefas](#) do Amazon Elastic Container Service (Amazon ECS).
- Use [perfis do IAM](#) para aplicações em execução nas instâncias do Amazon EC2.
- Use [credenciais temporárias](#) ou variáveis de ambiente para aplicativos disponíveis para usuários de fora da organização.

Tópicos

- [Usando o arquivo de AWS credenciais compartilhado](#)
- [Usando a SDK Store \(somente Windows\)](#)

Usando o arquivo de AWS credenciais compartilhado

(Assegure-se de revisar os [avisos e orientações importantes para credenciais](#).)

Uma forma de fornecer credenciais para seus aplicativos é criar perfis no arquivo de credenciais compartilhado da AWS e, em seguida, armazenar as credenciais nesses perfis. Esse arquivo pode ser usado pelo outro AWS SDKs. Ele também pode ser usado pelo [AWS CLI](#), [AWS Tools for Windows PowerShell](#), pelo e pelos AWS kits de ferramentas do [Visual Studio](#) e do [VS Code](#). [JetBrains](#)

Warning

Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como [Centro de Identidade do AWS IAM](#).

Note

As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais sobre credenciais de curto e longo prazo, consulte [Outras formas de autenticação no Guia](#) de referência de ferramentas AWS SDKs e ferramentas.

Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito em [Configure a autenticação do SDK](#).

Informações gerais

Por padrão, o arquivo de AWS credenciais compartilhadas está localizado no `.aws` diretório dentro do seu diretório inicial e é nomeado `credentials`; ou seja, `~/.aws/credentials` (Linux ou macOS) `%USERPROFILE%\aws\credentials` ou (Windows). Para obter informações sobre locais alternativos, consulte [Localização dos arquivos compartilhados](#) no [Guia de referência de ferramentas AWS SDKs e ferramentas](#). Também consulte [Acessar credenciais e perfis em um aplicativo](#).

O arquivo de AWS credenciais compartilhado é um arquivo de texto simples e segue um determinado formato. Para obter informações sobre o formato dos arquivos de AWS credenciais, consulte [Formato do arquivo de credenciais no Guia](#) de Referência de Ferramentas AWS SDKs e Ferramentas.

Você pode gerenciar os perfis no arquivo de AWS credenciais compartilhadas de várias maneiras.

- Use qualquer editor de texto para criar e atualizar o arquivo de AWS credenciais compartilhado.

- Use o [Amazon.Runtime.CredentialManagement](#) namespace da AWS SDK para .NET API, conforme mostrado posteriormente neste tópico.
- Use comandos e procedimentos para o [Ferramentas da AWS para PowerShell](#) e os AWS kits de ferramentas para [Visual Studio](#) e [VS Code](#). [JetBrains](#)
- Usar comandos do [AWS CLI](#); por exemplo, `aws configure set aws_access_key_id` e `aws configure set aws_secret_access_key`.

Exemplos de gerenciamento de perfil

As seções a seguir mostram exemplos de perfis no arquivo de AWS credenciais compartilhado. Alguns dos exemplos mostram o resultado, que pode ser obtido através de qualquer um dos métodos de gerenciamento de credenciais descritos anteriormente. Outros exemplos mostram como usar um método específico.

O perfil padrão.

O arquivo de AWS credenciais compartilhado quase sempre terá um perfil chamado default. É aqui que ele AWS SDK para .NET procura as credenciais se nenhum outro perfil estiver definido.

O perfil [default] geralmente é parecido com o mostrado a seguir.

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

Criar um perfil de forma programática

Este exemplo mostra como criar um perfil e salvá-lo programaticamente no arquivo de AWS credenciais compartilhadas. Ele usa as seguintes classes do [Amazon.Runtime.CredentialManagement](#) namespace: [CredentialProfileOptions](#), [CredentialProfile](#), e [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

Códigos como esse geralmente não deveriam estar em seu aplicativo. Se você inclui-lo em seu aplicativo, tome as devidas precauções para garantir que as chaves de texto simples não possam ser vistas no código, na rede ou mesmo na memória do computador.

A seguir está o perfil que foi criado por este exemplo.

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

Atualizar um perfil existente de forma programática

Este exemplo mostra como atualizar de forma programática o perfil que foi criado anteriormente. Ele usa as seguintes classes do [Amazon.Runtime.CredentialManagement](#) namespace: e. [CredentialProfileSharedCredentialsFile](#) Ele também usa a [RegionEndpoint](#) classe do namespace [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);
    }
}
```

A seguir está o perfil atualizado.

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

Note

Você também pode definir a AWS região em outros locais e usando outros métodos. Para obter mais informações, consulte [Configurar a região da AWS](#).

Usando a SDK Store (somente Windows)

(Assegure-se de revisar os [avisos e diretrizes importantes](#).)

No Windows, a SDK Store é outro local para criar perfis e armazenar credenciais criptografadas para seu AWS SDK para .NET aplicativo. Ela está localizada em %USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json. Você pode usar a SDK Store durante o desenvolvimento como alternativa ao [arquivo de credenciais compartilhado da AWS](#).

Warning

Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como [Centro de Identidade do AWS IAM](#).

 Note

As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais sobre credenciais de curto e longo prazo, consulte [Outras formas de autenticação no Guia de referência de ferramentas AWS SDKs e ferramentas](#).

Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito em [Configure a autenticação do SDK](#).

Informações gerais

A SDK Store oferece os seguintes benefícios:

- As credenciais na SDK Store são criptografadas, e a SDK Store reside no diretório base do usuário. Isso limita o risco de exposição acidental das credenciais.
- A SDK Store também oferece credenciais para o [AWS Tools for Windows PowerShell](#) e o [AWS Toolkit for Visual Studio](#).

Os perfis da SDK Store são específicos para um determinado usuário em um host particular. Não é possível copiá-los para outros hosts ou outros usuários. Isso significa que não é possível reutilizar os perfis da SDK Store que estiverem na máquina de desenvolvimento para outros hosts ou outras máquinas de desenvolvedores. Isso também significa que você não pode usar perfis da SDK Store em aplicativos de produção.

Gerencie os perfis na SDK Store das seguintes formas:

- Usar a interface gráfica de usuário (GUI) no [AWS Toolkit for Visual Studio](#).
- Use o [Amazon.Runtime.CredentialManagement](#) namespace da AWS SDK para .NET API, conforme mostrado posteriormente neste tópico.
- Usar comandos do [AWS Tools for Windows PowerShell](#); por exemplo, `Set-AWSCredential` e `Remove-AWSCredentialProfile`.

Exemplos de gerenciamento de perfil

Os exemplos a seguir mostram como criar e atualizar um perfil de forma programática na SDK Store.

Criar um perfil de forma programática

Este exemplo mostra como criar um perfil e salvá-lo na SDK Store de forma programática.

Ele usa as seguintes classes do [Amazon.Runtime.CredentialManagement](#) namespace:

[CredentialProfileOptions](#), [CredentialProfile](#), e [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

Códigos como esse geralmente não deveriam estar em seu aplicativo. Se ele estiver incluído em seu aplicativo, tome as devidas precauções para garantir que as chaves de texto simples não possam ser vistas no código, na rede ou mesmo na memória do computador.

A seguir está o perfil que foi criado por este exemplo.

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

Atualizar um perfil existente de forma programática

Este exemplo mostra como atualizar de forma programática o perfil que foi criado anteriormente.

Ele usa as seguintes classes do [Amazon.Runtime.CredentialManagement](#) namespace:

[CredentialProfile](#) e [Net SDKCredentials](#) File. Ele também usa a [RegionEndpoint](#) classe do namespace [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);
    }
}
```

A seguir está o perfil atualizado.

```
"[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

Você também pode definir a AWS região em outros locais e usando outros métodos. Para obter mais informações, consulte [Configurar a região da AWS](#).

Recursos da AWS SDK para .NET

Esta seção fornece informações sobre os atributos do AWS SDK para .NET que talvez você precise considerar ao criar seus aplicativos.

Certifique-se de ter [configurado seu projeto](#) primeiro.

Consulte informações sobre como desenvolver software para serviços da AWS específicos com exemplos de código em [Trabalhe com AWS serviços](#). Consulte mais exemplos de código em [SDK para .NET exemplos de código](#).

Tópicos

- [AWSAPIs assíncronas da para .NET](#)
- [Novas tentativas e tempos limite](#)
- [Paginadores](#)
- [Observabilidade](#)
- [Ferramentas adicionais](#)

AWSAPIs assíncronas da para .NET

O AWS SDK para .NET usa o Padrão Assíncrono Baseado em Tarefas (TAP) para sua implementação assíncrona. Para saber mais sobre o TAP, consulte [Padrão assíncrono baseado em tarefas \(TAP\)](#) em docs.microsoft.com.

Este tópico fornece uma visão geral de como usar o TAP em suas chamadas para clientes do serviços da AWS.

Os métodos assíncronos na API do AWS SDK para .NET são operações baseadas na classe Task ou na classe Task<TResult>. Consulte docs.microsoft.com para obter informações sobre essas classes: [classe Task](#), [classe Task<TResult>](#).

Quando esses métodos de API são chamados em seu código, eles devem ser chamados em uma função declarada com a palavra-chave async, conforme mostrado no exemplo a seguir.

```
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;
}
```

Conforme mostrado no trecho de código anterior, o escopo preferencial para a declaração `async` é a função `Main`. Definir esse escopo `async` garante que todas as chamadas para clientes de serviço da AWS sejam obrigatoriamente assíncronas. Se não puder declarar que `Main` seja assíncrono por algum motivo, você poderá usar a palavra-chave `async` em outras funções de `Main` e, em seguida, chamar os métodos da API a partir daí, conforme mostrado no exemplo a seguir.

```
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;
}
```

Observe a sintaxe especial `Task<>` que é necessária em `Main` quando você usa esse padrão. Além disso, você deve usar o membro **Result** da resposta para obter os dados.

Você pode ver exemplos completos de chamadas assíncronas para clientes AWS de serviço na seção [Faça um tour rápido](#) ([Aplicativo simples para várias plataformas](#) e [Aplicativo simples baseado no Windows](#)) e em [Exemplos de código com orientação](#).

Novas tentativas e tempos limite

O AWS SDK para .NET permite que você configure o número de novas tentativas e os valores de tempo limite para solicitações HTTP aos AWS serviços. Se os valores padrão para novas tentativas e tempos limite não forem apropriados, ajuste-os conforme os seus requisitos específicos, mas é importante entender como essa ação afetará o comportamento do aplicativo.

Para determinar quais valores usar para tentativas e tempos limite, considere o seguinte:

- Como o aplicativo AWS SDK para .NET e seu aplicativo devem responder quando a conectividade de rede se degrada ou um AWS serviço está inacessível? Deseja que a chamada falhe rapidamente, ou é apropriado que a chamada continue tentando em seu nome?
- O seu aplicativo é um aplicativo ou site voltado para o usuário e que deve ser responsivo, ou ele é um trabalho de processamento em segundo plano com maior tolerância a latências?
- O aplicativo é implantado em uma rede confiável com baixa latência ou é implantado em um local remoto com conectividade incerta?

Novas tentativas

Visão geral do

Eles AWS SDK para .NET podem repetir solicitações que falham devido à limitação do servidor ou à queda de conexões. Há duas propriedades das classes de configuração de serviço que você pode usar para especificar o comportamento de novas tentativas de um cliente de serviço. As classes de configuração de serviço herdam essas propriedades do abstrato [Amazon.Runtime.ClientConfig](#) classe da [referência da AWS SDK para .NET API](#):

- `RetryMode` [especifica um dos três modos de repetição, definidos no Amazon.Runtime.RequestRetryMode](#) enumeração.

O valor padrão do seu aplicativo pode ser controlado usando a variável de `AWS_RETRY_MODE` ambiente ou a configuração `retry_mode` no arquivo de configuração compartilhado AWS .

- `MaxErrorRetry` especifica o número de novas tentativas permitidas no nível do cliente de serviço; o SDK executa novas tentativas da operação pelo número especificado de vezes antes de causar uma falha e gerar uma exceção.

O valor padrão do seu aplicativo pode ser controlado usando a variável de `AWS_MAX_ATTEMPTS` ambiente ou a configuração `max_attempts` no arquivo de AWS configuração compartilhado.

As descrições detalhadas dessas propriedades podem ser encontradas no resumo [Amazon.Runtime.ClientConfig](#) classe da [Referência da AWS SDK para .NET API](#). Cada valor de `RetryMode` corresponde, por padrão, a um valor específico de `MaxErrorRetry`, conforme mostrado na tabela a seguir.

RetryMode	Correspondente MaxErrorRetry (Amazon DynamoDB)	Correspondente MaxErrorRetry (todos os outros)
Legacy	10	4
Standard	10	2
Adaptativo (experimental)	10	2

Comportamento

Quando seu aplicativo é iniciado

Quando seu aplicativo é iniciado, os valores padrão para `RetryMode` e `MaxErrorRetry` são configurados pelo SDK. Esses valores padrão são usados quando você cria um cliente de serviço, a menos que você especifique outros valores.

- Se as propriedades não estiverem definidas em seu ambiente, o padrão para `RetryMode` será configurado como `Legacy` e o padrão para `MaxErrorRetry` será configurado com o valor correspondente da tabela anterior.
- Se o modo de novas tentativas tiver sido definido em seu ambiente, esse valor será usado como padrão para `RetryMode`. O padrão para `MaxErrorRetry` é configurado com o valor correspondente da tabela anterior, a menos que o valor para o máximo de erros também tenha sido definido em seu ambiente (descrito a seguir).

- Se o valor para o máximo de erros tiver sido definido em seu ambiente, esse valor será usado como padrão para `MaxErrorRetry`. O Amazon DynamoDB é a exceção a essa regra; o valor padrão do DynamoDB para `MaxErrorRetry` é sempre o valor da tabela anterior.

À medida que seu aplicativo é executado

Ao criar um cliente de serviço, você pode usar os valores padrão para `RetryMode` e `MaxErrorRetry`, conforme descrito anteriormente, ou pode especificar outros valores. Para especificar outros valores, crie e inclua um objeto de configuração de serviço, como [AmazonDynamoDBConfig](#) ou [Amazon](#), `SQSConfig` ao criar o cliente de serviço.

Não é possível alterar os valores de um cliente de serviço depois que ele tiver sido criado.

Considerações

Quando ocorre uma nova tentativa, a latência da solicitação é aumentada. Configure as tentativas com base nos limites de latência total da solicitação e taxas de erros do aplicativo.

Tempos limite

AWS SDK para .NET Isso permite que você configure os tempos limite de solicitação no nível do cliente de serviço e por chamada de método. Há dois mecanismos para configurar tempos limite, que serão abordados nas seções subsequentes:

- Se você estiver usando [chamadas assíncronas](#), poderá utilizar o parâmetro `CancellationToken` do método.
- Se você estiver usando [chamadas síncronas no .NET Framework](#), poderá usar as `ReadWriteTimeout` propriedades `Timeout` e do abstrato [Amazon.Runtime.ClientConfig](#) classe.

Usar o parâmetro **CancellationToken** para tempos limite

O AWS SDK para .NET permite que você configure tempos limite de solicitação em chamadas assíncronas usando o parâmetro. `CancellationToken` O trecho de código a seguir mostra um exemplo. O código lançará `System.Threading.Tasks.TaskCanceledException` se a solicitação não for concluída em 10 segundos.

```
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);
}
```

Usar as propriedades **Timeout** e **ReadWriteTimeout** para tempos limite

Note

A propriedade `Timeout` não afeta as chamadas assíncronas. Se você estiver usando chamadas assíncronas, consulte [Usar o parâmetro `CancellationToken` para tempos limite](#) em vez disso.

O AWS SDK para .NET permite que você configure os valores de tempo limite da solicitação e do tempo read/write limite do soquete no nível do cliente de serviço. Esses valores são especificados nas `ReadWriteTimeout` propriedades `Timeout` e do resumo [Amazon.Runtime.ClientConfig](#) classe. Esses valores são transmitidos como `ReadWriteTimeout` propriedades `Timeout` e dos [HttpRequest](#) objetos criados pelo objeto cliente do AWS serviço. Por padrão, o valor `Timeout` é 100 segundos e o valor `ReadWriteTimeout` é 300 segundos.

Quando a sua rede estiver com latência alta, ou existirem condições que gerem uma nova tentativa para uma operação, utilizar valores de tempo limite longos e um alto número de tentativas pode fazer com que algumas operações do SDK pareçam indiferentes.

 Note

A versão do AWS SDK para .NET que tem como alvo a biblioteca de classes portátil (PCL) usa a [HttpClient](#) classe em vez da `HttpWebRequest` classe e oferece suporte somente à propriedade [Timeout](#).

Veja a seguir as exceções aos valores de tempo limite padrão. Estes valores são substituídos ao definir explicitamente os valores de tempo limite.

- `Timeout` e `ReadWriteTimeout` são definidos com os valores máximos se o método que está sendo chamado fizer upload de um stream, como [AmazonS3Client.PutObjectAsync\(\)](#), [cliente Amazon S3.UploadPartAsync\(\)](#), [AmazonGlacierClient.UploadArchiveAsync\(\)](#) e assim por diante.
- As versões do AWS SDK para .NET conjunto do .NET Framework têm como alvo os valores máximos `ReadWriteTimeout` para todos os objetos `Timeout` e clientes do [AmazonS3](#). [AmazonGlacierClient](#)
- As versões do AWS SDK para .NET que têm como alvo a biblioteca de classes portátil (PCL) e o .NET Core são definidas como o valor máximo `Timeout` para todos os objetos e clientes do [AmazonS3](#). [AmazonGlacierClient](#)

O exemplo a seguir mostra como especificar o modo de repetição padrão, no máximo 3 tentativas, um tempo limite de 10 segundos e um read/write tempo limite de 10 segundos (se aplicável). O construtor [AmazonS3Client](#) recebe um objeto [AmazonS3Config](#).

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

Paginadores

Alguns serviços da AWS coletam e armazenam uma grande quantidade de dados, que você pode recuperar usando as chamadas de API do AWS SDK para .NET. Se a quantidade de dados que você deseja recuperar se tornar muito grande para uma única chamada de API, você poderá dividir os resultados em partes mais gerenciáveis usando a paginação.

Para permitir que você realize a paginação, os objetos de solicitação e resposta de muitos clientes de serviço no SDK fornecem um token de continuação (normalmente intitulado `NextToken`). Alguns desses clientes de serviço também fornecem paginadores.

Os paginadores permitem que você evite a sobrecarga do token de continuação, que pode envolver loops, variáveis de estado, múltiplas chamadas de API e assim por diante. Ao usar um paginador, você pode recuperar dados de um serviço da AWS através de uma única linha de código, a declaração de um loop `foreach`. Se forem necessárias múltiplas chamadas de API para recuperar os dados, o paginador lidará com isso para você.

Onde posso encontrar paginadores?

Nem todos os serviços oferecem paginadores. Uma forma de determinar se um serviço fornece um paginador para uma API específica é examinar a definição de uma classe de cliente de serviço na [Referência de API do AWS SDK para .NET](#).

Por exemplo, se você examinar a definição da classe [AmazonCloudWatchLogsClient](#), verá uma propriedade `Paginators`. Essa é a propriedade que fornece um paginador para o Amazon CloudWatch Logs.

O que os paginadores me oferecem?

Os paginadores contêm propriedades que permitem que você veja as respostas completas. Eles também costumam conter uma ou mais propriedades que permitem acessar as partes mais interessantes das respostas, que chamaremos de resultados-chave.

Por exemplo, no `AmazonCloudWatchLogsClient` mencionado anteriormente, o objeto `Paginator` contém uma propriedade `Responses` com o objeto [DescribeLogGroupsResponse](#) completo da chamada de API. Essa propriedade `Responses` contém, entre outras coisas, uma coleção dos grupos de logs.

O objeto `Paginator` também contém um resultado chave chamado `LogGroups`. Essa propriedade contém apenas a parte da resposta dos grupos de logs. Ter esse resultados-chaves permite que você reduza e simplifique seu código em muitas circunstâncias.

Paginação síncrona em comparação à assíncrona

Os paginadores fornecem mecanismos síncronos e assíncronos para paginação. A paginação síncrona está disponível em projetos .NET Framework 4.7.2 (ou posterior). A paginação assíncrona está disponível em projetos do .NET Core (.NET Core 3.1, .NET 5 etc.).

Como as operações assíncronas e o .NET Core são recomendadas, o exemplo a seguir mostra a paginação assíncrona. As informações sobre como realizar as mesmas tarefas usando paginação síncrona e o .NET Framework 4.7.2 (ou posterior) são mostradas após o exemplo em [Considerações adicionais sobre paginadores](#).

Exemplo

O exemplo a seguir mostra como usar o AWS SDK para .NET para exibir uma lista de grupos de logs. Para fins de comparação, o exemplo mostra como fazer isso com e sem paginadores. Antes de examinar o código completo, mostrado posteriormente, considere os trechos a seguir.

Como obter grupos de logs do CloudWatch sem paginadores

```
// 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));
```

Como obter grupos de logs do CloudWatch usando paginadores

```
// 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);  
}
```

Os resultados desses dois trechos são exatamente os mesmos, então a vantagem de usar paginadores pode ser vista claramente.

Note

Antes de tentar criar e executar o código completo, certifique-se de ter [configurado seu ambiente e projeto](#).

Talvez você também precise do pacote NuGet [Microsoft.Bcl.AsyncInterfaces](#) porque paginadores assíncronos usam a interface `IAsyncEnumerable`.

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

Pacotes NuGet:

- [AWSSDK.CloudWatch](#)

Elementos de programação:

- Namespace [Amazon.CloudWatch](#)
 - Classe [AmazonCloudWatchLogsClient](#)
- Namespace [Amazon.CloudWatchLogs.Model](#)
 - Classe [DescribeLogGroupsRequest](#)
 - Classe [DescribeLogGroupsResponse](#)
 - Classe [LogGroup](#)

Código completo

```
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}");
        }
    }
}

```

```
}  
}  
}  
}  
}
```

Considerações adicionais sobre paginadores

- Os paginadores não podem ser usados mais de uma vez

Se precisar dos resultados de um paginador da AWS específico em vários locais em seu código, você não deve usar um objeto paginador mais de uma vez. Em vez disso, crie um novo paginador sempre que precisar dele. Esse conceito é mostrado no exemplo de código anterior no método `DisplayLogGroupsWithPaginators`.

- Paginação síncrona

A paginação síncrona está disponível para projetos .NET Framework 4.7.2 (ou posterior).

Warning

A partir de 15 de agosto de 2024, o AWS SDK para .NET encerrará o suporte para o .NET Framework 3.5 e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

Para ver isso, crie um projeto do .NET Framework 4.7.2 (ou posterior) e copie o código anterior nele. Em seguida, basta remover a palavra-chave `await` das duas chamadas do paginador `foreach`, conforme mostrado no exemplo a seguir.

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

Crie e execute o projeto para ver os mesmos resultados que você viu com a paginação assíncrona.

Observabilidade

Observabilidade é a medida em que o estado atual de um sistema pode ser identificado com base nos dados que ele emite. Os dados emitidos são chamados, com frequência, de telemetria.

Eles AWS SDK para .NET podem fornecer dois sinais, métricas e rastreamentos de telemetria comuns, além de registros. Você pode conectar um [TelemetryProvider](#) para enviar dados de telemetria para um back-end de observabilidade (como a [AWS X-RayAmazon CloudWatch](#)) e, em seguida, agir de acordo com eles.

Por padrão, os sinais de telemetria permanecem desabilitados no SDK. Este tópico explica como habilitar e configurar a saída de telemetria.

Recursos adicionais do

Para acessar mais informações sobre como habilitar e usar a observabilidade, consulte os seguintes recursos:

- [OpenTelemetry](#)
- A postagem do blog [Enhancing Observability in the AWS SDK para .NET with](#). OpenTelemetry
- A postagem do blog [anunciando a disponibilidade geral das AWS OpenTelemetry bibliotecas.NET](#).
- [Exportadores para OpenTelemetry](#)
- Para exemplos de observabilidade no Ferramentas da AWS para PowerShell, consulte [Observabilidade no Guia de Ferramentas do PowerShell Usuário](#).

Configurar um **TelemetryProvider**

Você pode configurar um `TelemetryProvider` em sua aplicação globalmente para todos os clientes de serviço ou para clientes individuais, conforme mostrado nos exemplos a seguir. A seção [the section called “Provedores de telemetria”](#) contém informações sobre implementações de telemetria, incluindo informações sobre implementações fornecidas com o SDK.

Configurar o provedor de telemetria global padrão

Por padrão, todo cliente de serviço tenta usar o provedor de telemetria disponível globalmente. Dessa forma, é possível definir o provedor uma vez e todos os clientes o usarão. Isso deve ser feito apenas uma vez, antes de criar qualquer cliente de serviço.

O trecho de código a seguir mostra como definir o provedor de telemetria global. Depois, ele cria um cliente de serviço Amazon S3 e tenta realizar uma operação que falha. O código adiciona rastreamento e métricas à aplicação. Esse código usa os seguintes NuGet pacotes: `OpenTelemetry.Exporter.Console` `OpenTelemetry.Instrumentation.AWS` e.

 Note

Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar `AWSSDK.SSO` `AWSSDK.SSO0IDC` e.

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

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

Sdk.CreateMeterProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .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();
```

Configurar um provedor de telemetria para um cliente de serviço específico

É possível configurar um cliente de serviço individual com um provedor de telemetria específico (diferente do global). Para isso, use a classe `TelemetryProvider` do objeto `Config` de um construtor de cliente de serviço. Por exemplo, consulte [AmazonS3Config](#) e procure a propriedade `TelemetryProvider`. Consulte [the section called “Provedores de telemetria”](#) para acessar informações sobre implementações personalizadas de telemetria.

Tópicos

- [Métricas](#)
- [Provedores de telemetria](#)

Métricas

A tabela a seguir lista as métricas de telemetria emitidas pelo SDK. [Configure um provedor de telemetria](#) para tornar as métricas observáveis.

Quais métricas são emitidas?

Nome da métrica	Unidades	Tipo	Atributos	Descrição
<code>client.call.duration</code>	s	Histograma	<code>rpc.service</code> , <code>rpc.method</code>	Duração geral da chamada (incluindo novas tentativas e tempo para enviar ou receber a solicitação e o corpo da resposta).
<code>client.uptime</code>	s	Histograma	<code>rpc.service</code>	A quantidade de tempo desde a criação de um cliente.
<code>client.call.attempts</code>	{attempt}	Monoton Counter	<code>rpc.service</code> , <code>rpc.method</code>	O número de tentativas de uma operação individual.
<code>client.call.errors</code>	{error}	Monoton Counter	<code>rpc.service</code> , <code>rpc.method</code> , <code>exception.type</code>	O número de erros de uma operação.

Nome da métrica	Unidades	Tipo	Atributos	Descrição
client.call.attempt_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para se conectar ao serviço, enviar a solicitação e recuperar o código de status HTTP e os cabeçalhos (incluindo o tempo de espera para envio).
client.call.resolve_endpoint_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para resolver um endpoint (resolvedor de endpoint, não DNS) para a solicitação.
client.call.serialization_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para serializar o corpo da mensagem.
client.call.deserialization_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para desserializar o corpo da mensagem.
client.call.auth.signing_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para assinar uma solicitação
client.call.auth.resolve_identity_duration	s	Histograma	rpc.service, rpc.method	O tempo necessário para adquirir uma identidade (como credenciais da AWS ou um token de portador) de um provedor de identidade
client.http.bytes_sent	Por	Monoton Counter	server.address	O número total de bytes enviados pelo cliente HTTP
client.http.bytes_received	Por	Monoton Counter	server.address	O número total de bytes recebidos pelo cliente HTTP

Veja a seguir as descrições das colunas.

- Nome da métrica: o nome da métrica emitida.
- Unidades: a unidade de medida da métrica. As unidades são fornecidas na notação [UCUM](#) com distinção entre maiúsculas e minúsculas (“c/s”).

- Tipo: o tipo de instrumento usado para capturar a métrica.
- Atributos: o conjunto de atributos (dimensões) emitidos com a métrica.
- Descrição: descrição do que a métrica está medindo.

Provedores de telemetria

O SDK oferece uma implementação do [OpenTelemetry](#) como provedor de telemetria, descrita na [próxima seção](#).

Se você tem requisitos específicos de telemetria, já tem uma solução em mente ou precisa de um controle refinado sobre como os dados de telemetria são capturados e processados, você também pode implementar seu próprio provedor.

Registre sua própria implementação com a classe [TelemetryProvider](#). Veja a seguir um exemplo simples de como registrar seus próprios `TracerProvider` e `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());
```

Tópicos

- [Configurar o provedor OpenTelemetry de telemetria baseado](#)

Configurar o provedor OpenTelemetry de telemetria baseado

AWS SDK para .NET Isso inclui a implementação de um provedor de telemetria OpenTelemetry baseado. Para acessar detalhes sobre como definir esse provedor como o provedor global de telemetria, consulte [Configurar um TelemetryProvider](#). Para usar esse provedor de telemetria, você precisa dos seguintes recursos em seu projeto:

- O pacote [OpenTelemetry NuGet .Instrumentation.aws](#).
- Um exportador de telemetria, como OTLP ou Console. Para obter mais informações, consulte [Exportadores](#) na OpenTelemetry documentação.

A OpenTelemetry implementação incluída no SDK pode ser configurada para reduzir a quantidade de rastreamento de solicitações, credenciais e compactação HTTPS. Para fazer isso, defina a opção `SuppressDownstreamInstrumentation` como `true`, semelhante à seguinte:

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

Para obter informações adicionais sobre esse provedor, consulte a postagem do blog [Enhancing Observability in the AWS SDK para .NET with](#). OpenTelemetry

Ferramentas adicionais

A seguir estão algumas ferramentas adicionais que você pode usar para facilitar o trabalho de desenvolvimento, implantação e manutenção de seus aplicativos .NET.

Ferramenta de implantação da AWS

Depois de desenvolver seu aplicativo .NET Core nativo de nuvem em uma máquina de desenvolvimento, você pode usar a Ferramenta de Implantação da AWS para a CLI do .NET para implantar seu aplicativo com mais facilidade em AWS.

Para obter mais informações, consulte [Implantar aplicativos na AWS](#).

Estrutura de processamento de mensagens para .NET da AWS

Se você estiver usando serviços como Amazon SQS, Amazon SNS ou Amazon EventBridge, poderá utilizar a Estrutura de processamento de mensagens para .NET da AWS. Para obter mais informações, consulte [Estrutura de processamento de mensagens para .NET da AWS](#).

Integrações ao .NET Aspire

Você pode aproveitar as vantagens das integrações ao .NET Aspire para melhorar o ciclo interno de desenvolvimento. Para obter mais informações, consulte [Integrar a AWS ao .NET Aspire no AWS SDK para .NET](#).

Autenticação e autorização avançadas com o AWS SDK para .NET

Os tópicos desta seção fornecem informações sobre técnicas avançadas de autenticação e autorização em seu aplicativo AWS SDK para .NET.

Tópicos

- [Autenticação única com o AWS SDK para .NET](#)

Autenticação única com o AWS SDK para .NET

O Centro de Identidade do AWS IAM é um serviço de login único (SSO) na nuvem que facilita o gerenciamento centralizado do acesso de SSO a todas as suas Contas da AWS e a aplicativos de nuvem. Para obter detalhes completos, consulte o [Guia do usuário do Centro de Identidade do IAM](#).

Se você não estiver familiarizado com a forma como um SDK interage com o Centro de Identidade do IAM, consulte as informações a seguir.

Padrão de interação de alto nível

Em um alto nível, os SDKs interagem com o Centro de Identidade do IAM de maneira semelhante ao seguinte padrão:

1. O Centro de Identidade do IAM é configurado, normalmente por meio do [console do Centro de Identidade do IAM](#) e um usuário de SSO é convidado a participar.
2. O arquivo `config` da AWS compartilhado no computador do usuário é atualizado com informações de SSO.
3. O usuário faz login por meio do Centro de Identidade do IAM e recebe credenciais de curto prazo para as permissões do AWS Identity and Access Management (IAM) que foram configuradas para ele. Esse login pode ser iniciado por meio de uma ferramenta que não é do SDK, como a AWS CLI, ou de forma programática por meio de um aplicativo .NET.
4. O usuário continua fazendo seu trabalho. Quando outros aplicativos que usam SSO são executados, não é necessário logar novamente para abrir os aplicativos.

O restante deste tópico fornece informações de referência para configuração e uso de Centro de Identidade do AWS IAM. Ele fornece informações complementares e mais avançadas do que a

configuração básica do SSO em [Configure a autenticação do SDK](#). Se você é iniciante no SSO na AWS, talvez queira ver esse tópico primeiro para obter informações fundamentais e, em seguida, os seguintes tutoriais para ver o SSO em ação:

- [Tutorial: somente aplicativo .NET](#)
- [Tutorial: AWS CLI e aplicativo.NET](#)

Este tópico contém as seguintes seções:

- [Pré-requisitos](#)
- [Como configurar um perfil de SSO](#)
- [Como gerar e usar de tokens de SSO](#)
- [Recursos adicionais](#)
- [Tutoriais](#)

Pré-requisitos

Antes de usar o Centro de Identidade do IAM, você deve realizar determinadas tarefas, como escolher uma fonte de identidade e configurar as Contas da AWS e aplicativos relevantes. Para obter informações adicionais, consulte:

- Para obter informações gerais sobre essas tarefas, consulte [Conceitos básicos](#) no Guia do usuário do Centro de Identidade do IAM
- Para obter exemplos de tarefas específicas, consulte a lista de tutoriais no final deste tópico. Contudo, certifique-se de revisar as informações neste tópico antes de experimentar os tutoriais.

Como configurar um perfil de SSO

Depois que o Centro de Identidade do IAM for [configurado](#) na Conta da AWS relevante, um perfil nomeado para SSO deve ser adicionado ao arquivo `config` da AWS compartilhado do usuário. Esse perfil é usado para se conectar ao [portal de acesso da AWS](#), que retorna credenciais de curto prazo para as permissões do IAM que foram configuradas para o usuário.

O arquivo `config` compartilhado geralmente é nomeado `%USERPROFILE%\aws\config` no Windows e `~/.aws/config` no Linux e no macOS. Você pode usar seu editor de texto preferido para adicionar um novo perfil para SSO. Como alternativa, você pode usar o comando `aws`

configure `sso`. Para obter mais informações sobre esse comando, consulte [Como configurar a CLI da AWS para usar o Centro de identidade do IAM](#) no Guia do usuário do AWS Command Line Interface.

O URL é semelhante ao seguinte:

```
[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 = SSOReadOnlyRole
```

As configurações do novo perfil estão definidas abaixo. As duas primeiras configurações definem o portal de acesso da AWS. As outras duas configurações são um par que, juntas, definem as permissões que foram configuradas para um usuário. Todas as quatro configurações são obrigatórias.

sso_start_url

O URL que aponta para o [portal de acesso da AWS](#) da organização. Para encontrar esse valor, abra o [console do Centro de identidade do IAM](#), selecione Configurações e encontre URL do portal.

sso_region

A Região da AWS que contém o host do portal de acesso da . Essa é a região que foi selecionada quando você ativou o Centro de identidade do IAM. Ela pode ser diferente das regiões que você usa para outras tarefas.

Para obter uma lista completa das Regiões da AWS e seus códigos, consulte [Endpoints regionais](#) na Referência geral da Amazon Web Services.

sso_account_id

O ID de uma Conta da AWS que foi adicionada por meio do serviço AWS Organizations. Para ver a lista de contas disponíveis, acesse o [console do IAM Identity Center](#) e abra a página de Contas da AWS. O ID da conta que você escolher para essa configuração corresponderá ao valor que você planeja atribuir à configuração do `sso_role_name`, que é mostrado a seguir.

sso_role_name

O nome de um conjunto de permissões do Centro de identidade do IAM. Esse conjunto de permissões define as permissões que um usuário recebe por meio do Centro de identidade do IAM.

O procedimento a seguir é uma forma de encontrar o valor dessa configuração.

1. Acesse o [console do Centro de Identidade do IAM](#) e habilite a página das Contas da AWS.
2. Escolha uma conta para exibir os detalhes. A conta que você escolher será aquela que contém o usuário ou grupo de SSO ao qual você deseja conceder permissões de SSO.
3. Veja a lista de usuários e grupos atribuídos à conta e encontre o usuário ou grupo de interesse. O conjunto de permissões que você especifica na configuração do `sso_role_name` é um dos conjuntos associados a esse usuário ou grupo.

Ao dar um valor a essa configuração, use o nome do conjunto de permissões, não o nome do recurso da Amazon (ARN).

Os conjuntos de permissões têm políticas do IAM e políticas de permissões personalizadas anexadas a eles. Para obter mais informações, consulte [Conjuntos de permissões](#) no Guia do usuário do Centro de Identidade do IAM.

Como gerar e usar de tokens de SSO

Para usar o SSO, o usuário deve primeiro gerar um token temporário e depois usar esse token para acessar os aplicativos e recursos da AWS apropriados. É possível usar os métodos a seguir para gerar e usar esses tokens temporários:

- Crie aplicativos .NET que gerem primeiro um token, se necessário, e depois use o token.
- Gere um token com a AWS CLI e, em seguida, use o token em aplicativos .NET.

Esses métodos estão descritos nas seções a seguir e demonstrados nos [tutoriais](#).

Important

Seu aplicativo deve fazer referência aos seguintes pacotes NuGet para que a resolução de SSO possa funcionar:

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

A falha em referenciar esses pacotes resultará em uma exceção de tempo de execução.

Somente aplicativo .NET

Esta seção mostra como criar um aplicativo .NET que gera um token SSO temporário, se necessário, e depois usa esse token. Para obter um tutorial completo desse processo, consulte [Tutorial para SSO usando somente aplicativos .NET](#).

Gere e use um token SSO de forma programática

Além de usar a AWS CLI, você também pode gerar um token SSO de forma programática.

Para fazer isso, seu aplicativo cria um objeto [AWSCredentials](#) para o perfil SSO, que carrega credenciais temporárias, se houver alguma disponível. Em seguida, seu aplicativo deve converter o objeto `AWSCredentials` em um objeto [SSOAWSCredentials](#) e definir algumas propriedades de [Opções](#), incluindo um método de retorno de chamada usado para solicitar ao usuário informações de login, se necessário.

Esse método é mostrado no seguinte trecho de código.

Important

Seu aplicativo deve fazer referência aos seguintes pacotes NuGet para que a resolução de SSO possa funcionar:

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

A falha em referenciar esses pacotes resultará em uma exceção de tempo de execução.

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

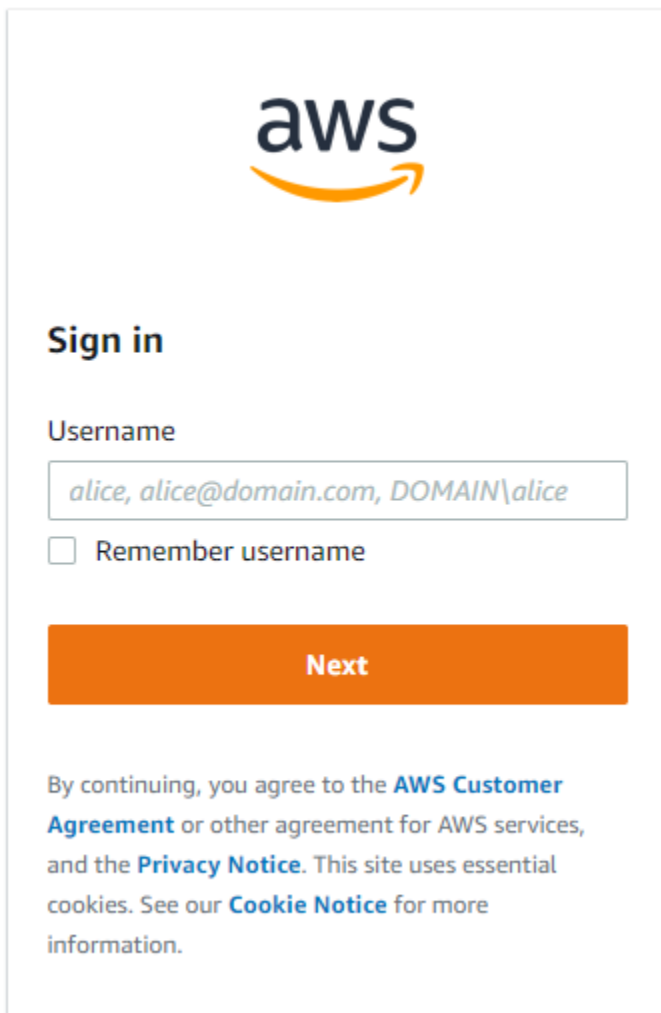
```
var chain = new CredentialProfileStoreChain();
if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
    throw new Exception("Failed to find the my-sso-profile profile");

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;
}
```

Se um token de SSO apropriado não estiver disponível, a janela padrão do navegador será aberta e a página de login apropriada será aberta. Por exemplo, se você estiver usando o Centro de identidade do IAM como fonte de identidade, o usuário verá uma página de login semelhante à seguinte:



The image shows 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". Below the checkbox is an orange button with the text "Next". 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

A string que você fornece para `SSOAWSCredentials.Options.ClientName` não pode ter espaços. Se a string tiver espaços, você receberá uma exceção de runtime.

[Tutorial para SSO usando somente aplicativos .NET](#)

AWS CLI e aplicativo .NET

Esta seção mostra como gerar um token SSO temporário usando a AWS CLI e como usar esse token em um aplicativo. Para obter um tutorial completo desse processo, consulte [Tutorial para SSO usando os aplicativos AWS CLI e .NET](#).

Gere um token de SSO usando o AWS CLI

Além de gerar um token de SSO temporário de forma programática, você usa a AWS CLI para gerar o token. O procedimento a seguir mostra como obter essas informações.

Depois que o usuário cria um perfil habilitado para SSO, conforme mostrado na [seção anterior](#), ele executa o comando `aws sso login` a partir da AWS CLI. Ele deve ter certeza de incluir o parâmetro `--profile` com o nome do perfil habilitado para SSO. Isso é mostrado no exemplo a seguir:

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

Se o usuário quiser gerar um novo token temporário após a expiração do atual, ele poderá executar o mesmo comando novamente.

Use o token SSO gerado em um aplicativo .NET

As informações a seguir mostram como usar um token temporário que já foi gerado.

Important

Seu aplicativo deve fazer referência aos seguintes pacotes NuGet para que a resolução de SSO possa funcionar:

- AWSSDK.SSO
- AWSSDK.SSO0IDC

A falha em referenciar esses pacotes resultará em uma exceção de tempo de execução.

Seu aplicativo cria um objeto [AWSCredentials](#) para o perfil SSO, que carrega as credenciais temporárias geradas anteriormente pela AWS CLI. Isso é semelhante aos métodos mostrados em [Acessar credenciais e perfis em um aplicativo](#) e tem o seguinte formato:

```
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;  
}
```

O objeto `AWSCredentials` é então passado para o construtor de um cliente de serviço. Por exemplo:

```
var S3Client_SSO = new AmazonS3Client(LoadSsoCredentials());
```

Note

Não é necessário usar `AWSCredentials` para carregar credenciais temporárias se seu aplicativo tiver sido criado para usar o perfil [default] para SSO. Nesse caso, o aplicativo pode criar clientes AWS de serviço sem parâmetros, semelhantes a `"var client = new AmazonS3Client();"` .

[Tutorial para SSO usando os aplicativos AWS CLI e.NET](#)

Recursos adicionais

Para obter ajuda adicional, consulte os seguintes recursos.

- [O que é o Centro de identidade do IAM?](#)
- [Configurar a AWS CLI para usar o Centro de identidade do IAM](#)
- [Usar as credenciais do Centro de identidade do IAM no AWS Toolkit for Visual Studio](#)

Tutoriais

Tópicos

- [Tutorial para SSO usando somente aplicativos .NET](#)
- [Tutorial para SSO usando os aplicativos AWS CLI e.NET](#)

Tutorial para SSO usando somente aplicativos .NET

Este tutorial mostra como habilitar o SSO para um aplicativo básico e um usuário de SSO de teste. Ele configura o aplicativo para gerar um token SSO temporário de forma programática em vez de [usar a AWS CLI](#).

Este tutorial mostra uma pequena parte da funcionalidade de SSO no AWS SDK para .NET. Para obter detalhes completos sobre o uso do IAM Identity Center com o AWS SDK para .NET, consulte o tópico com [informações básicas](#). Nesse tópico, consulte especialmente a descrição de alto nível desse cenário na subseção chamada [Somente aplicativo .NET](#).

Note

Várias das etapas deste tutorial ajudam você a configurar serviços como AWS Organizations o IAM Identity Center. Se já realizou essa configuração ou se está interessado apenas no código, você pode pular para a seção com o [código de exemplo](#).

Pré-requisitos

- Configure seu ambiente de desenvolvimento, caso ainda não tenha feito isso. Isso é descrito em seções como [Instale e configure seu conjunto de ferramentas](#) e [Conceitos básicos](#).
- Identifique ou crie pelo menos um Conta da AWS que você possa usar para testar o SSO. Para os fins deste tutorial, isso é chamado de Conta da AWS de teste ou simplesmente conta de teste.
- Identifique um usuário de SSO que possa testar o SSO para você. Essa é uma pessoa que usará o SSO e os aplicativos básicos que você criar. Neste tutorial, essa pessoa pode ser você (o desenvolvedor) ou outra pessoa. Também recomendamos uma configuração na qual o usuário do SSO trabalhe em um computador que não esteja em seu ambiente de desenvolvimento. Contudo, isso não é estritamente necessário.
- O computador do usuário do SSO deve ter uma estrutura do .NET instalada que seja compatível com a que você usou para configurar seu ambiente de desenvolvimento.

Configurar AWS

Esta seção mostra como configurar vários AWS serviços para este tutorial.

Para realizar essa configuração, primeiro entre no teste Conta da AWS como administrador. Faça o seguinte:

Amazon S3

Acesse o [console do Amazon S3](#) e adicione alguns buckets inócuos. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses buckets.

AWS IAM

Acesse o [console do IAM](#) e adicione alguns usuários do IAM. Se você conceder permissões aos usuários do IAM, limite as permissões a algumas permissões inócuas somente para leitura. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses usuários do IAM.

AWS Organizations

Acesse o [console do AWS Organizations](#) e ative o Organizations. Para obter mais informações, consulte [Criar uma organização](#) no [Guia do usuário do AWS Organizations](#).

Essa ação adiciona o teste Conta da AWS à organização como a conta de gerenciamento. Se você tiver contas de teste adicionais, poderá convidá-las para participar da organização, mas isso não é necessário para este tutorial.

Centro de Identidade do IAM

Acesse o [console do Centro de Identidade do IAM](#) e habilite o SSO. Execute a verificação por e-mail, se necessário. Para obter mais informações, consulte [Habilitar o Centro de Identidade do IAM](#) no [Guia do usuário do Centro de Identidade do IAM](#).

Depois, execute a configuração a seguir.

Configure o Centro de Identidade do IAM

1. Vá para a página Configurações. Procure a “URL do portal de acesso” e registre o valor para uso posterior na configuração do `sso_start_url`.
2. No banner do Console de gerenciamento da AWS, procure o Região da AWS que foi definido quando você ativou o SSO. Esse é o menu suspenso à esquerda do Conta da AWS ID. Registre o código da região para uso posterior na configuração da `sso_region`. Esse código será semelhante a `us-east-1`.
3. Crie um usuário de SSO da seguinte forma:
 - a. Acesse a página Usuários.
 - b. Escolha Adicionar usuário e insira o nome de usuário, endereço de e-mail, nome e sobrenome do usuário. Em seguida, escolha Próximo.
 - c. Escolha Avançar na página para grupos, revise as informações e escolha Adicionar usuário.
4. Crie um grupo da seguinte forma:

- a. Acesse a página Grupos.
 - b. Escolha Criar grupo e insira o nome do grupo e a descrição do grupo.
 - c. Na seção Adicionar usuários ao grupo, selecione o usuário de SSO de teste que você criou anteriormente. Selecione Criar grupo.
5. Crie um conjunto de permissões da seguinte forma:
- a. Acesse a página Conjuntos de permissões e escolha Criar conjunto de permissões.
 - b. Em Tipo de conjunto de permissões, selecione Conjunto de permissões personalizado e escolha Avançar.
 - c. Abra a política em linha e insira a seguinte política:

JSON

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

- d. Para este tutorial, insira `SSOReadOnlyRole` como o nome do conjunto de permissões. Adicione uma descrição, se quiser, e escolha Avançar.
 - e. Revise as informações e escolha Criar.
 - f. Registre o nome do conjunto de permissões para uso posterior na configuração do `sso_role_name`.
6. Acesse a página de AWS contas e escolha a AWS conta que você adicionou à organização anteriormente.
7. Na seção Visão geral dessa página, encontre o ID da conta e registre-a para uso posterior na configuração do `sso_account_id`.

8. Escolha a guia Usuários e grupos e, em seguida, escolha Atribuir usuários ou grupos.
9. Na página Atribuir usuários e grupos, escolha a guia Grupos, selecione o grupo que você criou anteriormente e escolha Avançar.
10. Selecione o conjunto de permissões que você criou anteriormente e escolha Avançar e, em seguida, escolha Enviar. A configuração leva alguns instantes.

Como criar aplicativos de exemplo

Crie os aplicativos a seguir. Eles serão executados no computador do usuário do SSO.

Listar buckets do Amazon S3

Inclui NuGet pacotes `AWSSDK.S3` e `AWSSDK.SS00IDC` além de `AWSSDK.S3` `AWSSDK.SecurityToken` e.

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

// 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.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($"\\nSSO 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-SS0-App";
        ssoCredentials.Options.SsoVerificationCallback = args =>
        {
            // Launch a browser window that prompts the SS0 user to complete an SS0
login.
            // This method is only invoked if the session doesn't already have a
valid SS0 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;
    }
}
}

```

Listar usuários do IAM

Inclui NuGet pacotes `AWSSDK.SSO` e, `AWSSDK.SSO0IDC` além de `AWSSDK.IdentityManagement` `AWSSDK.SecurityToken` e.

```

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;
    }
}
}

```

Além de exibir listas de buckets do Amazon S3 e usuários do IAM, esses aplicativos exibem o ARN de identidade do usuário para o perfil habilitado para SSO, que está neste tutorial. `my-sso-profile`

Esses aplicativos realizam tarefas de login com SSO fornecendo um método de retorno de chamada na propriedade [Options](#) de um objeto. [SSOAWSCredentials](#)

Instrua o usuário do SSO

Peça ao usuário do SSO que verifique seu e-mail e aceite o convite do SSO. Será solicitado que ele defina uma senha. A mensagem pode levar alguns minutos para chegar à caixa de entrada do usuário do SSO.

Forneça ao usuário do SSO os aplicativos criados anteriormente.

Em seguida, peça ao usuário do SSO que faça o seguinte:

1. Se a pasta que contém o AWS config arquivo compartilhado não existir, crie-a. Se a pasta existir e tiver uma subpasta chamada `.sso`, exclua essa subpasta.

A localização dessa pasta geralmente é `%USERPROFILE%\ .aws` no Windows e `~/ .aws` no Linux e no macOS.

2. Crie um AWS config arquivo compartilhado nessa pasta, se necessário, e adicione um perfil a ele da seguinte maneira:

```
[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. Execute o aplicativo do Amazon S3.
4. Na página de login na web resultante, faça login. Use o nome de usuário da mensagem de convite e a senha que foi criada em resposta à mensagem.
5. Quando o login estiver concluído, o aplicativo exibirá a lista de buckets do S3.
6. Execute o aplicativo do IAM. O aplicativo exibe a lista de usuários do IAM. Isso é verdade mesmo que um segundo login não tenha sido realizado. O aplicativo IAM usa o token temporário que foi criado anteriormente.

Limpeza

Se você não quiser manter os recursos que criou durante este tutorial, apague-os. Eles podem ser AWS recursos ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Tutorial para SSO usando os aplicativos AWS CLI e .NET

Este tutorial mostra como habilitar o SSO para um aplicativo .NET básico e um usuário de SSO de teste. Ele usa o AWS CLI para gerar um token SSO temporário em vez de [gerá-lo programaticamente](#).

Este tutorial mostra uma pequena parte da funcionalidade de SSO no AWS SDK para .NET. Para obter detalhes completos sobre o uso do IAM Identity Center com o AWS SDK para .NET, consulte o tópico com [informações básicas](#). Nesse tópico, consulte especialmente a descrição de alto nível desse cenário na subseção chamada [AWS CLI e aplicativo .NET](#).

Note

Várias das etapas deste tutorial ajudam você a configurar serviços como AWS Organizations o IAM Identity Center. Se já realizou essas configurações ou se está interessado apenas no código, você pode pular para a seção com o [código de exemplo](#).

Pré-requisitos

- Configure seu ambiente de desenvolvimento, caso ainda não tenha feito isso. Isso é descrito em seções como [Instale e configure seu conjunto de ferramentas](#) e [Conceitos básicos](#).
- Identifique ou crie pelo menos um Conta da AWS que você possa usar para testar o SSO. Para os fins deste tutorial, isso é chamado de Conta da AWS de teste ou simplesmente conta de teste.
- Identifique um usuário de SSO que possa testar o SSO para você. Essa é uma pessoa que usará o SSO e os aplicativos básicos que você criar. Neste tutorial, essa pessoa pode ser você (o desenvolvedor) ou outra pessoa. Também recomendamos uma configuração na qual o usuário do SSO trabalhe em um computador que não esteja em seu ambiente de desenvolvimento. Contudo, isso não é estritamente necessário.
- O computador do usuário do SSO deve ter uma estrutura do .NET instalada que seja compatível com a que você usou para configurar seu ambiente de desenvolvimento.
- Certifique-se de que a AWS CLI versão 2 esteja [instalada](#) no computador do usuário do SSO. Você pode verificar isso executando `aws --version` em um prompt de comando ou terminal.

Configurar AWS

Esta seção mostra como configurar vários AWS serviços para este tutorial.

Para realizar essa configuração, primeiro entre no teste Conta da AWS como administrador. Faça o seguinte:

Amazon S3

Acesse o [console do Amazon S3](#) e adicione alguns buckets inócuos. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses buckets.

AWS IAM

Acesse o [console do IAM](#) e adicione alguns usuários do IAM. Se você conceder permissões aos usuários do IAM, limite as permissões a algumas permissões inócuas somente para leitura. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses usuários do IAM.

AWS Organizations

Acesse o [console do AWS Organizations](#) e ative o Organizations. Para obter mais informações, consulte [Criar uma organização](#) no [Guia do usuário do AWS Organizations](#).

Essa ação adiciona o teste Conta da AWS à organização como a conta de gerenciamento. Se você tiver contas de teste adicionais, poderá convidá-las para participar da organização, mas isso não é necessário para este tutorial.

Centro de Identidade do IAM

Acesse o [console do Centro de Identidade do IAM](#) e habilite o SSO. Execute a verificação por e-mail, se necessário. Para obter mais informações, consulte [Habilitar o Centro de Identidade do IAM](#) no [Guia do usuário do Centro de Identidade do IAM](#).

Depois, execute a configuração a seguir.

Configure o Centro de Identidade do IAM

1. Vá para a página Configurações. Procure a “URL do portal de acesso” e registre o valor para uso posterior na configuração do `sso_start_url`.
2. No banner do Console de gerenciamento da AWS, procure o Região da AWS que foi definido quando você ativou o SSO. Esse é o menu suspenso à esquerda do Conta da AWS ID. Registre o código da região para uso posterior na configuração da `sso_region`. Esse código será semelhante a `us-east-1`.
3. Crie um usuário de SSO da seguinte forma:
 - a. Acesse a página Usuários.
 - b. Escolha Adicionar usuário e insira o nome de usuário, endereço de e-mail, nome e sobrenome do usuário. Em seguida, escolha Próximo.

- c. Escolha Avançar na página para grupos, revise as informações e escolha Adicionar usuário.
4. Crie um grupo da seguinte forma:
 - a. Acesse a página Grupos.
 - b. Escolha Criar grupo e insira o nome do grupo e a descrição do grupo.
 - c. Na seção Adicionar usuários ao grupo, selecione o usuário de SSO de teste que você criou anteriormente. Selecione Criar grupo.
5. Crie um conjunto de permissões da seguinte forma:
 - a. Acesse a página Conjuntos de permissões e escolha Criar conjunto de permissões.
 - b. Em Tipo de conjunto de permissões, selecione Conjunto de permissões personalizado e escolha Avançar.
 - c. Abra a política em linha e insira a seguinte política:

JSON

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

- d. Para este tutorial, insira `SSOReadOnlyRole` como o nome do conjunto de permissões. Adicione uma descrição, se quiser, e escolha Avançar.
 - e. Revise as informações e escolha Criar.
 - f. Registre o nome do conjunto de permissões para uso posterior na configuração do `sso_role_name`.

6. Acesse a página de AWS contas e escolha a AWS conta que você adicionou à organização anteriormente.
7. Na seção Visão geral dessa página, encontre o ID da conta e registre-a para uso posterior na configuração do `sso_account_id`.
8. Escolha a guia Usuários e grupos e, em seguida, escolha Atribuir usuários ou grupos.
9. Na página Atribuir usuários e grupos, escolha a guia Grupos, selecione o grupo que você criou anteriormente e escolha Avançar.
10. Selecione o conjunto de permissões que você criou anteriormente e escolha Avançar e, em seguida, escolha Enviar. A configuração leva alguns instantes.

Como criar aplicativos de exemplo

Crie os aplicativos a seguir. Eles serão executados no computador do usuário do SSO.

Listar buckets do Amazon S3

Inclui NuGet pacotes `AWSSDK.S3` e `AWSSDK.SSO0IDC` além de `AWSSDK.S3` `AWSSDK.SecurityToken` e.

```
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 SS0Example.S3.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 S3 buckets.
    // The S3 client is created using the SSO 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;
        }
    }
}

```

Listar usuários do IAM

Inclui NuGet pacotes `AWSSDK.SSO` e `AWSSDK.SSO0IDC` além de `AWSSDK.IdentityManagement` `AWSSDK.SecurityToken` e.

```

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 SSOExample.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;
        }
    }
}

```


Além de exibir listas de buckets do Amazon S3 e usuários do IAM, esses aplicativos exibem o ARN de identidade do usuário para o perfil habilitado para SSO, que está neste tutorial. `my-sso-profile`

Instrua o usuário do SSO

Peça ao usuário do SSO que verifique seu e-mail e aceite o convite do SSO. Será solicitado que ele defina uma senha. A mensagem pode levar alguns minutos para chegar à caixa de entrada do usuário do SSO.

Forneça ao usuário do SSO os aplicativos criados anteriormente.

Em seguida, peça ao usuário do SSO que faça o seguinte:

1. Se a pasta que contém o AWS config arquivo compartilhado não existir, crie-a. Se a pasta existir e tiver uma subpasta chamada `.sso`, exclua essa subpasta.

A localização dessa pasta geralmente é `%USERPROFILE%\ .aws` no Windows e `~/ .aws` no Linux e no macOS.

2. Crie um AWS config arquivo compartilhado nessa pasta, se necessário, e adicione um perfil a ele da seguinte maneira:

```
[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. Execute o aplicativo do Amazon S3. Uma exceção de tempo de execução é exibida.
4. Execute o seguinte AWS CLI comando:

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

5. Na página de login na web resultante, faça login. Use o nome de usuário da mensagem de convite e a senha que foi criada em resposta à mensagem.
6. Execute o aplicativo do Amazon S3 novamente. Agora, o aplicativo exibe a lista de buckets do S3.

7. Execute o aplicativo do IAM. O aplicativo exibe a lista de usuários do IAM. Isso é verdade mesmo que um segundo login não tenha sido realizado. O aplicativo IAM usa o token temporário que foi criado anteriormente.

Limpeza

Se você não quiser manter os recursos que criou durante este tutorial, apague-os. Eles podem ser AWS recursos ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Implantar aplicativos na AWS

Depois de desenvolver seu aplicativo ou serviço do .NET Core nativo de nuvem em uma máquina de desenvolvimento, você vai desejar implantá-lo na AWS. Você pode fazer isso usando o Console de gerenciamento da AWS ou determinados serviços, como CloudFormation ou AWS Cloud Development Kit (AWS CDK). Você também pode usar ferramentas da AWS que foram criadas para fins de implantação. Usando essas ferramentas, você pode fazer o seguinte:

Implantar a partir da CLI do .NET

Você pode usar as seguintes ferramentas da AWS para a CLI do .NET implantar seus aplicativos em AWS:

- [Ferramenta de implantação da AWS para a CLI do .NET](#): oferece suporte a implantações no [AWS App Runner](#), [Amazon Elastic Container Service \(Amazon ECS\)](#) e [AWS Elastic Beanstalk](#).
- [Ferramentas do AWS Lambda para a CLI do .NET](#): oferece suporte para implantações de projetos AWS Lambda.

Implante a partir dos kits de ferramentas IDE

Você pode usar os kits de ferramentas da AWS para implantar seus aplicativos diretamente do IDE de sua escolha:

- [AWS Toolkit for Visual Studio](#)

Note

O atributo “Publicar na AWS” no kit de ferramentas expõe a mesma funcionalidade da Ferramenta de Implantação da AWS para a CLI do .NET. Para saber mais, acesse [Publicar na AWS](#) no Guia do usuário do AWS Toolkit for Visual Studio.

- [AWS Toolkit for JetBrains](#)

Consulte [Trabalhar com aplicativos sem servidor da AWS](#) e [Trabalhar com AWS App Runner](#).

- [AWS Toolkit for VS Code da](#)

Consulte [Como trabalhar com aplicativos sem servidor](#) e [Como usar AWS App Runner](#).

- [Kit de ferramentas da AWS para Azure DevOps](#)

Casos de uso

As seções a seguir contêm cenários de casos de uso para determinados tipos de aplicações, incluindo informações sobre como você usaria a CLI do .NET para implantar essas aplicações.

- [Aplicativos ASP.NET Core:](#)
- [Aplicativos do .NET Console](#)
- [Aplicativos do Blazor WebAssembly](#)
- [AWS Lambda projetos](#)

Aplicativos ASP.NET Core:

A [ferramenta de implantação da AWS](#) para a CLI do .NET o ajuda a implantar seus aplicativos do ASP.NET e serve como orientação em um processo de implantação. É uma ferramenta interativa para a CLI do .NET que ajuda a implantar aplicativos .NET com conhecimento mínimo sobre a AWS.

A ferramenta de implantação tem os seguintes recursos:

- Recomendações de computação para seu aplicativo: obtenha as recomendações de computação e saiba qual computação da AWS é mais adequada para seu aplicativo.
- Geração de Dockerfile: a ferramenta gera um Dockerfile, se necessário, ou usa um Dockerfile existente.
- Empacotamento e implantação automáticos: a ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de implantação AWS CDK gerado e implanta seu aplicativo na computação da AWS escolhida.
- Implantações reproduzíveis e compartilháveis: você pode gerar e modificar projetos de implantação AWS CDK de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado de AWS CDK para .NET: a ferramenta o ajuda a aprender gradualmente as ferramentas da AWS subjacentes nas quais ele se baseia, como o AWS CDK.

A [Ferramenta de implantação da AWS](#) oferece suporte à implantação dos aplicativos ASP.NET Core aos seguintes serviços da AWS:

- [Amazon ECS Service](#) usando [AWS Fargate](#): oferece suporte a implantações de aplicativos web no Amazon Elastic Container Service (Amazon ECS) com poder computacional gerenciado por um mecanismo de computação sem servidor AWS Fargate.
- [AWS App Runner](#): oferece suporte a implantações em um serviço totalmente gerenciado que facilita para os desenvolvedores a implantação de aplicativos web em contêineres e APIs em escala. Não é necessária nenhuma experiência prévia em infraestrutura.
- [AWS Elastic Beanstalk](#): oferece suporte a implantações em um serviço que facilita para os desenvolvedores a implantação de aplicativos web e APIs em um ambiente gerenciado e em escala. Não é necessária nenhuma experiência prévia em infraestrutura.

Para saber mais, consulte a [visão geral da ferramenta](#). Para começar a partir dali, navegue até Documentação, Conceitos básicos e escolha [Como instalar](#) para receber instruções de instalação.

Aplicativos do .NET Console

A [ferramenta de implantação da AWS](#) para a CLI do .NET ajuda você a implantar seus aplicativos do .NET Console como um serviço ou uma tarefa agendada como uma imagem de contêiner no Linux e serve como orientação em um processo de implantação. Se seu aplicativo não tiver um Dockerfile, a ferramenta o gerará automaticamente. Caso contrário, um Dockerfile existente será usado.

A ferramenta de implantação tem os seguintes recursos:

- Recomendações de computação para seu aplicativo: obtenha as recomendações de computação e saiba qual computação da AWS é mais adequada para seu aplicativo.
- Geração de Dockerfile: a ferramenta gera um Dockerfile, se necessário, ou usa um Dockerfile existente.
- Empacotamento e implantação automáticos: a ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de implantação AWS CDK gerado e implanta seu aplicativo na computação da AWS escolhida.
- Implantações reproduzíveis e compartilháveis: você pode gerar e modificar projetos de implantação AWS CDK de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado de AWS CDK para .NET: a ferramenta o ajuda a aprender gradualmente as ferramentas da AWS subjacentes nas quais ele se baseia, como o AWS CDK.

A [Ferramenta de implantação da AWS](#) oferece suporte à implantação aos seguintes serviços da AWS:

- [Amazon ECS Service](#) usando [AWS Fargate](#): oferece suporte a implantações de aplicativos .NET como um serviço (por exemplo, um processador em segundo plano) no Amazon Elastic Container Service (Amazon ECS) com poder computacional gerenciado por um mecanismo de computação sem servidor AWS Fargate.
- [Tarefa programada do Amazon ECS](#) usando [AWS Fargate](#): oferece suporte a implantações de aplicativos .NET como uma tarefa programada (por exemplo, processo de fim de dia) no Amazon ECS com poder computacional gerenciado por um mecanismo de computação sem servidor AWS Fargate.

Para saber mais, consulte a [visão geral da ferramenta](#). Para começar a partir dali, navegue até Documentação, Conceitos Básicos e escolha [Como instalar](#) para obter instruções de instalação.

Aplicativos do Blazor WebAssembly

A [ferramenta de implantação da AWS](#) para a CLI do .NET o ajuda a hospedar seu aplicativo Blazor WebAssembly no Amazon S3, usando o Amazon CloudFront para fornecimento na rede de conteúdo. Seu aplicativo é implantado em um bucket do S3 para hospedagem na web. A ferramenta cria e configura um bucket S3 e, em seguida, carrega seu aplicativo Blazor no bucket.

A ferramenta de implantação tem os seguintes recursos:

- Empacotamento e implantação automáticos: a ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de implantação AWS CDK gerado e implanta seu aplicativo na computação da AWS escolhida.
- Implantações reproduzíveis e compartilháveis: você pode gerar e modificar projetos de implantação AWS CDK de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado de AWS CDK para .NET: a ferramenta o ajuda a aprender gradualmente as ferramentas da AWS subjacentes nas quais ele se baseia, como o AWS CDK.

Para saber mais, consulte a [visão geral da ferramenta](#). Para começar a partir dali, navegue até Documentação, Conceitos básicos e escolha [Como instalar](#) para receber instruções de instalação.

AWS Lambda projetos

AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. Ele executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais. Para obter mais informações sobre o Lambda, consulte [O que é o AWS Lambda?](#) no Guia do AWS Lambda desenvolvedor.

Você pode implantar funções do Lambda usando a interface da linha de comandos (CLI) do .NET.

Tópicos

- [Pré-requisitos](#)
- [Comandos do Lambda disponíveis](#)
- [Etapas para implantar](#)

Pré-requisitos

Antes de começar a usar a CLI do .NET para implantar funções do Lambda, você deve atender os seguintes pré-requisitos:

- Confirme se você tem a CLI do .NET instalada. Por exemplo: `dotnet --version`. Se necessário, acesse <https://dotnet.microsoft.com/download> para instalá-lo.
- Configure a CLI do .NET para trabalhar com o Lambda. Para obter uma descrição de como fazer isso, consulte [CLI do .NET Core](#) no Guia do Desenvolvedor do AWS Lambda . Nesse procedimento, o seguinte é o comando de implantação:

```
dotnet lambda deploy-function MyFunction --function-role role
```

Se você não tiver certeza de como criar um perfil do IAM para este exercício, não inclua a parte `--function-role role`. A ferramenta ajudará você a criar um novo perfil.

Comandos do Lambda disponíveis

Para listar os comandos do Lambda que estão disponíveis por meio da CLI do .NET, abra um prompt de comando ou terminal e digite `dotnet lambda --help`. A saída desse comando será semelhante a:

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>
```

A saída lista todos os comandos que estão disponíveis atualmente.

Etapas para implantar

As instruções a seguir pressupõem que você tenha criado um AWS Lambda projeto.NET. Para fins desse procedimento, o projeto é denominado DotNetCoreLambdaTest.

1. Abra um prompt de comando ou terminal e navegue até a pasta que contém o arquivo de projeto .NET Lambda.
2. Insira `dotnet lambda deploy-function`.
3. Se solicitado, insira a AWS Região (a região na qual sua função Lambda será implantada).
4. Quando solicitado, digite o nome da função a ser implantada, por exemplo, DotNetCoreLambdaTest. Pode ser o nome de uma função que já exista na sua Conta da AWS ou de uma que ainda não tenha sido implantada ali.
5. Quando solicitado, selecione ou crie o perfil do IAM que o Lambda vai pressupor ao executar a função.

Após a conclusão ter sido bem-sucedida, a mensagem Nova função do Lambda criada é exibida.

```
Executing publish command
```

```
...
```

```
(etc.)
```

```
New Lambda function created
```


Se você implantar uma função que já existe em sua conta, a função de implantação solicitará somente a Região da AWS (se necessário). Nesse caso, a saída do comando termina com `Updating code for existing function`.

Depois que a função do Lambda for implantada, ela estará pronta para ser usada. Para obter mais informações, consulte [exemplos de como usar o AWS Lambda](#).

O Lambda monitora automaticamente as funções do Lambda para você e relata métricas por meio da Amazon CloudWatch. Para monitorar e solucionar os problemas da sua função do Lambda, consulte [Monitoramento e solução de problemas de aplicativos Lambda](#).

Migre o seu projeto para o AWS SDK para .NET

Esta seção fornece informações sobre as tarefas de migração que podem se aplicar a você e instruções sobre como realizá-las.

Tópicos

- [Migrar para a versão 3 do AWS SDK para .NET](#)
- [Migrar para a versão 3.5 do AWS SDK para .NET](#)
- [Migrar para a versão 3.7 do AWS SDK para .NET](#)
- [Migração do .NET Standard 1.3](#)

Migrar para a versão 3 do AWS SDK para .NET

Este tópico descreve as alterações na versão 3 do AWS SDK para .NET e como migrar o seu código para esta versão do SDK.

Sobre as versões do AWS SDK para .NET

O AWS SDK para .NET, lançado originalmente em novembro de 2009, foi projetado para o .NET Framework 2.0. Desde o lançamento, o .NET foi aperfeiçoado com o .NET Framework 4.0 e o .NET Framework 4.5, e adicionou novas plataformas de destino: WinRT e Windows Phone.

AWS SDK para .NET versão 2 do foi atualizada para aproveitar os novos recursos da plataforma .NET e para atingir o WinRT e o Windows Phone.

AWS SDK para .NET versão 3 do foi atualizada para tornar os conjuntos modulares.

Redefinição da arquitetura para o SDK

Toda a versão 3 do AWS SDK para .NET está redefinida para ser modular. Agora, cada serviço está implementado em seu próprio conjunto, em vez de um único conjunto global. Não é mais necessário adicionar todo o AWS SDK para .NET ao aplicativo. Adicione conjuntos somente para os serviços da AWS utilizados pelo aplicativo.

Alterações que podem causar interrupções

As seções a seguir descrevem as alterações na versão 3 do AWS SDK para .NET.

AWSClientFactory removida

A classe `Amazon.AWSClientFactory` foi removida. Agora, para criar um cliente de serviço, use o construtor do cliente de serviço. Por exemplo, para criar um `AmazonEC2Client`:

```
var ec2Client = new Amazon.EC2.AmazonEC2Client();
```

Amazon.Runtime.AssumeRoleAWSCredentials removida

A classe `Amazon.Runtime.AssumeRoleAWSCredentials` foi removida pois estava em um namespace central, mas tinha uma dependência no AWS Security Token Service, e porque era obsoleta no SDK por algum tempo. No lugar, use a classe `Amazon.SecurityToken.AssumeRoleAWSCredentials`.

Método SetACL removido da S3Link

A classe `S3Link` faz parte do pacote `Amazon.DynamoDBv2` e é usada para armazenar objetos no Amazon S3 que são referências em um item do DynamoDB. Este é um atributo útil, mas não queríamos criar uma dependência de compilação no pacote `Amazon.S3` para o DynamoDB. Consequentemente, simplificamos os métodos `Amazon.S3` expostos da classe `S3Link`, substituindo o método `SetACL` pelo método `MakeS3ObjectPublic`. Para obter mais controle sobre a lista de controle de acesso (ACL) no objeto, use o pacote `Amazon.S3` diretamente.

Remoção de classes de resultados obsoletas

Para a maioria dos serviços do AWS SDK para .NET, as operações retornam um objeto de resposta que contém metadados para a operação, como o ID da solicitação e um objeto de resultado. Ter classes de resultados e respostas separadas era redundante e gerava linhas extras para os desenvolvedores. Na versão 2 do AWS SDK para .NET, colocamos todas as informações da classe de resultados na classe de respostas. Também marcamos as classes de resultados como obsoletas para desencorajar o seu uso. Na versão 3 do AWS SDK para .NET, removemos essas classes de resultados obsoletas para ajudar a reduzir o tamanho do SDK.

Alterações na seção AWS Config

É possível realizar a configuração avançada do AWS SDK para .NET por meio dos arquivos `App.config` ou `Web.config`. Faça isso por meio de uma seção de configuração da `<aws>` como a seguinte, que faz referência ao nome do conjunto do SDK.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Na versão 3 do AWS SDK para .NET, o conjunto AWSSDK não existe mais. Colocamos o código comum no conjunto AWSSDK.Core. Como resultado, será necessário alterar as referências ao conjunto AWSSDK nos arquivos App.config ou Web.config para o conjunto AWSSDK.Core, como mostrado a seguir.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Também é possível manipular as definições de configuração com a classe `Amazon.AWSConfigs`. Na versão 3 do AWS SDK para .NET, movemos as definições de configuração do DynamoDB da classe `Amazon.AWSConfigs` para a classe `Amazon.AWSConfigsDynamoDB`.

Migrar para a versão 3.5 do AWS SDK para .NET

A versão 3.5 do AWS SDK para .NET padroniza ainda mais a experiência do .NET fazendo a transição do suporte para todas as variações que não sejam do Framework do SDK para o [.NET Standard 2.0](#). Dependendo do ambiente e da base de código, para aproveitar os recursos da versão 3.5, talvez seja necessário executar certos trabalhos de migração.

Este tópico descreve as alterações na versão 3.5 e o possível trabalho que talvez seja necessário para migrar seu ambiente ou código da versão 3.

O que mudou na versão 3.5

Veja a seguir o que mudou ou não no AWS SDK para .NET versão 3.5.

.NET Framework e .NET Core

O suporte ao .NET Framework e ao .NET Core não foi alterado.

Xamarin

Os projetos Xamarin (novos e existentes) devem ter como destino o .NET Standard 2.0. Consulte [Suporte do .NET Standard 2.0 no Xamarin.Forms](#) e [Suporte à implementação do .NET](#).

Unity

Os aplicativos Unity devem ter como destino os perfis do .NET Standard 2.0 ou do .NET 4.x usando o Unity 2018.1 ou posterior. Para obter mais informações, consulte [Suporte a perfis do .NET](#). Além disso, se estiver usando IL2CPP para criar, você deverá desativar a remoção de código adicionando um arquivo link.xml conforme descrito em [Fazer referência ao AWS SDK para .NET Standard 2.0 do Unity, Xamarin ou UWP](#). Depois que transferir seu código para uma das bases de código recomendadas, seu aplicativo Unity poderá acessar todos os serviços oferecidos pelo SDK.

Como o Unity oferece suporte ao .NET Standard 2.0, o pacote AWSSDK.Core do SDK versão 3.5 não tem mais código específico do Unity, incluindo algumas funcionalidades de nível superior. Para proporcionar uma transição melhor, todo o código legado do Unity está disponível para referência no repositório [aws/aws-sdk-unity-net](#) do GitHub. Se encontrar uma funcionalidade ausente que afete seu uso da AWS com o Unity, você poderá registrar uma solicitação de atributo em <https://github.com/aws/dotnet/issues>.

Consulte também [Considerações especiais sobre o suporte ao Unity](#).

Plataforma Universal do Windows (UWP)

Dirija seu aplicativo UWP para a [versão 16299 ou posterior](#) (Fall Creators Update, versão 1709, lançada em outubro de 2017).

Windows Phone e Silverlight

A versão 3.5 do AWS SDK para .NET não oferece suporte a essas plataformas porque a Microsoft não está mais desenvolvendo-as ativamente. Para obter mais informações, consulte:

- [Fim do suporte ao Windows 10 Mobile](#)
- [Fim do suporte ao Silverlight](#)

Bibliotecas de classes portáteis legadas (PCLs baseadas em perfis)

Considere redirecionar sua biblioteca para o .NET Standard. Para obter mais informações, consulte [Comparação com bibliotecas de classes portáteis](#) da Microsoft.

Gerente do Amazon Cognito Sync e Gerente do Amazon Mobile Analytics

As abstrações de alto nível que facilitam o uso do Amazon Cognito Sync e do Amazon Mobile Analytics foram removidas da versão 3.5 do AWS SDK para .NET. AWS AppSync é o substituto preferencial para o Amazon Cognito Sync. O Amazon Pinpoint é o substituto preferencial do Amazon Mobile Analytics.

Se o seu código for afetado pela falta de código de biblioteca de nível superior para o AWS AppSync e o Amazon Pinpoint, você poderá registrar seu interesse em um ou ambos os seguintes chamados do GitHub: <https://github.com/aws/dotnet/issues/20> e <https://github.com/aws/dotnet/issues/19>. Também é possível obter as bibliotecas para o Gerenciador do Amazon Cognito Sync e o Gerenciador do Amazon Mobile Analytics nos seguintes repositórios do GitHub: [aws/amazon-cognito-sync-manager-net](#) e [aws/aws-mobile-analytics-manager-net](#).

Migrar código síncrono

A versão 3.5 do AWS SDK para .NET oferece suporte tanto para o .NET Framework quanto para o .NET Standard (por meio de versões do .NET Core, como o .NET core 3.1, .NET 5 e assim por diante). As variações do SDK que estão em conformidade com o .NET Standard fornecem somente métodos assíncronos, portanto, se você quiser aproveitar o .NET Standard, deverá alterar o código síncrono para que ele seja executado de forma assíncrona.

Os trechos de código a seguir mostram como alterar o código síncrono para código assíncrono. O código nesses trechos é usado para exibir o número de buckets do Amazon S3.

O código original chama [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}");
```

Para usar a versão 3.5 do SDK, chame [ListBucketsAsync](#).

```
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}");
```

Migrar para a versão 3.7 do AWS SDK para .NET

A partir da versão 3.7, o AWS SDK para .NET deixou de ser compatível com o .NET Standard 1.3.

Para obter informações sobre a migração do .NET Standard 1.3, consulte [Migração do .NET Standard 1.3](#).

Migração do .NET Standard 1.3

Em 27 de junho de 2019, a Microsoft [encerrou o suporte](#) para as versões .NET Core 1.0 e .NET Core 1.1. De acordo com esse anúncio, a AWS terminará o suporte para o .NET Standard 1.3 no AWS SDK para .NET em 31 de dezembro de 2020.

A AWS continuou a fornecer atualizações de serviço e correções de segurança no AWS SDK para .NET em relação ao .NET Standard 1.3 até 1º de outubro de 2020. Após essa data, o alvo .NET Standard 1.3 entrou no modo de Manutenção, o que significou que nenhuma nova atualização seria lançada; a AWS aplicou somente correções de bugs críticos e patches de segurança.

Em 31 de dezembro de 2020, o suporte para o .NET Standard 1.3 no AWS SDK para .NET chegou ao fim da vida útil. Após esta data, nenhuma correção de bugs ou patches de segurança foi aplicada. Artefatos criados com esse alvo continuam disponíveis para download no NuGet.

O que você precisa fazer

- Se estiver executando aplicativos usando .NET Framework, você não será afetado.
- Se você estiver executando aplicativos usando .NET Core 2.0 ou superior, você não será afetado.
- Se estiver executando aplicativos usando .NET Core 1.0 ou .NET Core 1.1, faça a migração dos seus aplicativos para uma versão mais recente do .NET Core seguindo as [instruções de migração da Microsoft](#). Recomendamos no mínimo .NET Core 3.1.
- Se estiver executando aplicativos críticos para os negócios que não podem ser atualizados neste momento, você poderá continuar usando sua versão atual do AWS SDK para .NET.

Se tiver dúvidas ou perguntas, [entre em contato com o AWS Support](#).

Trabalhe com AWS serviços no AWS SDK para .NET

As seções a seguir contêm exemplos, tutoriais, tarefas e guias que mostram como usar o AWS SDK para .NET para trabalhar com AWS serviços. Esses exemplos e tutoriais dependem de uma API que o AWS SDK para .NET fornece. Para ver quais classes e métodos estão disponíveis na API, consulte a [Referência de API do AWS SDK para .NET](#).

Se você é novo no AWS SDK para .NET, talvez queira conferir o [Faça um tour rápido](#) tópico primeiro. Ele fornece uma introdução ao SDK.

Você pode encontrar mais exemplos de código no Repositório de [exemplos de AWS código e no repositório awslabs](#) em. GitHub

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

- [Exemplos de código com orientação para o AWS SDK para .NET](#)
- [Usando AWS Lambda para serviço de computação](#)
- [Bibliotecas e estruturas de alto nível para o AWS SDK para .NET](#)
- [Suporte para outros serviços da AWS e configuração](#)

Exemplos de código com orientação para o AWS SDK para .NET

As seções a seguir contêm exemplos de código e fornecem orientação para os exemplos. Eles podem ajudá-lo a aprender como usar o AWS SDK para .NET para trabalhar com AWS serviços.

Se você é novo no AWS SDK para .NET, talvez queira conferir o [Faça um tour rápido](#) tópico primeiro. Ele fornece uma introdução ao SDK.

Antes de começar, assegure-se de [configurar o ambiente e o projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

- [Acessando CloudFormation com o AWS SDK para .NET](#)
- [Autenticar usuários com o Amazon Cognito](#)
- [Como usar os bancos de dados NoSQL Amazon DynamoDB](#)

- [Trabalhar com o Amazon EC2](#)
- [Acessando AWS Identity and Access Management \(IAM\) com o AWS SDK para .NET](#)
- [Usar o armazenamento para internet Amazon Simple Storage Service](#)
- [Enviar notificações da nuvem usando o Amazon Simple Notification Service](#)
- [Enviar mensagens usando o Amazon SQS](#)

Acessando CloudFormation com o AWS SDK para .NET

Os AWS SDK para .NET suportam [AWS CloudFormation](#), que criam e provisionam implantações de AWS infraestrutura de forma previsível e repetida.

APIs

Eles AWS SDK para .NET fornecem APIs para CloudFormation clientes. Eles APIs permitem que você trabalhe com CloudFormation recursos como modelos e pilhas. Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a [Referência da AWS SDK para .NET API](#) (e vá até “Amazon. CloudFormation”).

Eles AWS CloudFormation APIs são fornecidos pelo [AWSSDK. CloudFormation](#) pacote.

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

Tópicos

- [Listando AWS recursos usando AWS CloudFormation](#)

Listando AWS recursos usando AWS CloudFormation

Este exemplo mostra como usar o AWS SDK para .NET para listar os recursos em CloudFormation pilhas. O exemplo usa a API de baixo nível. O aplicativo não aceita argumentos, apenas reúne informações de todas as pilhas acessíveis às credenciais do usuário e, em seguida, exibe informações sobre essas pilhas.

Referências do SDK

NuGet pacotes:

- [AWSSDK.CloudFormation](#)

Elementos de programação:

- [Namespace Amazon. CloudFormation](#)

Classe [AmazonCloudFormationClient](#)

- [Namespace Amazon. CloudFormation.Modelo](#)

Classe [ICloudFormationPaginatorFactory. DescribeStacks](#)

Classe [DescribeStackResourcesRequest](#)

Classe [DescribeStackResourcesResponse](#)

Classe [Stack](#)

Classe [StackResource](#)

Classe [Tag](#)

```
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($"Stack: {stack.StackName}");
                Console.WriteLine($"  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;  
}  
}
```

Autenticar usuários com o Amazon Cognito

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

Usando o Amazon Cognito Identity, você pode criar identidades exclusivas para seus usuários e autenticá-los para acesso seguro aos seus AWS recursos, como o Amazon S3 ou o Amazon DynamoDB. O Amazon Cognito Identity é compatível com provedores de identidade públicos, como o Amazon, o Facebook e o Google, ou qualquer provedor compatível com o OpenID Connect, bem como identidades não autenticadas. O Amazon Cognito é compatível também com as [identidades autenticadas pelos desenvolvedores](#), que permitem registrar e autenticar usuários por meio do processo de autenticação de back-end, sem deixar de usar a Sincronização do Amazon Cognito para sincronizar os dados do usuário e acessar os recursos AWS .

Para obter mais informações sobre o [Amazon Cognito](#), consulte o [Amazon Cognito Developer Guide](#).

Os exemplos de código a seguir mostram como usar facilmente o Amazon Cognito Identity. O exemplo [Provedor de credenciais](#) mostra como criar e autenticar identidades de usuários. O [CognitoAuthentication biblioteca de extensão](#) exemplo mostra como usar a biblioteca de CognitoAuthentication extensões para autenticar grupos de usuários do Amazon Cognito.

Tópicos

- [Provedor de credenciais do Amazon Cognito](#)
- [Exemplos de bibliotecas CognitoAuthentication de extensões da Amazon](#)

Provedor de credenciais do Amazon Cognito

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, encontrado no [AWSSDK.CognitoIdentity](#) NuGetpackage, é um objeto de credenciais que usa o Amazon Cognito e AWS Security Token Service o AWS STS() para recuperar credenciais para fazer chamadas. AWS

A primeira etapa da configuração do `CognitoAWSCredentials` é criar um "grupo de identidades". Um grupo de identidades é um armazenamento de informações de identidades de usuários específico para sua conta. As informações podem ser recuperadas entre plataformas, dispositivos e sistemas operacionais de clientes, para que, se um usuário começar a usar o aplicativo em um telefone e depois mudar para um tablet, as informações mantidas do aplicativo ainda estejam disponíveis para esse usuário. É possível criar um novo banco de identidades no console do Amazon Cognito. Se você está usando o console, ele também oferece outras informações de que você precisa:

- Seu número de conta – um número de 12 dígitos, como 123456789012, que é exclusivo para sua conta.
- O ARN da função não autenticada – uma função que os usuários não autenticados assumirão. Por exemplo, essa função pode fornecer permissões somente leitura aos seus dados.
- O ARN da função autenticada – uma função que os usuários autenticados assumirão. Essa função pode fornecer permissões mais amplas aos seus dados.

Configurar o Cognito AWSCredentials

O código de exemplo a seguir mostra como configurar o `CognitoAWSCredentials`, que pode ser usado para fazer uma chamada para o Amazon S3 como usuário não autenticado. Isso permite que você faça chamadas com apenas uma quantidade mínima de dados necessária para autenticar o usuário. As permissões de usuários são controladas pela função para que seja possível configurar o acesso conforme necessário.

```
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();
}
```

Use AWS como um usuário não autenticado

O exemplo de código a seguir mostra como você pode começar a usar AWS como usuário não autenticado, depois se autenticar pelo Facebook e atualizar as credenciais para usar as credenciais do Facebook. Usando essa abordagem, você pode conceder capacidades diferentes a usuários autenticados por meio da função autenticada. Por exemplo, você pode ter um aplicativo de telefone que permite que os usuários visualizem conteúdo anonimamente, mas lhes permite publicar se eles estão conectados com um ou mais dos provedores configurados.

```
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();
}
```

O objeto `CognitoAWSCredentials` oferecerá ainda mais funcionalidades se você usá-lo com o `AmazonCognitoSyncClient` que faz parte do AWS SDK para .NET. Se você estiver usando `AmazonCognitoSyncClient` e `CognitoAWSCredentials`, não é necessário

especificar as propriedades `IdentityPoolId` e `IdentityId` ao fazer chamadas com o `AmazonCognitoSyncClient`. Essas propriedades são preenchidas automaticamente em `CognitoAWSCredentials`. O exemplo de código a seguir ilustra isso, bem como um evento que notifica você sempre que o `IdentityId` para `CognitoAWSCredentials` é alterado. O `IdentityId` pode ser alterado em alguns casos, como na alteração de um usuário não autenticado para um autenticado.

```
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 = "..."
    });
}
```

Exemplos de bibliotecas CognitoAuthentication de extensões da Amazon

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

A biblioteca `CognitoAuthentication` de extensões, encontrada em [Amazon.Extensions.CognitoAuthentication](#) NuGet pacote, simplifica o processo de autenticação dos grupos de usuários do Amazon Cognito para desenvolvedores do .NET Core e Xamarin. A biblioteca foi criada com base na API do provedor de identidade do Amazon Cognito para criar e enviar chamadas de API de autenticação de usuários.

Usando a biblioteca CognitoAuthentication de extensões

O Amazon Cognito tem alguns valores AuthFlow e ChallengeName integrados para que um fluxo de autenticação padrão valide o nome de usuário e a senha pelo protocolo SRP (Secure Remote Password). Para obter mais informações sobre o fluxo de autenticação, consulte [Amazon Cognito User Pool Authentication Flow \(Fluxo de autenticação de grupos de usuários do Amazon Cognito\)](#).

Os exemplos a seguir exigem estas instruções 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;
```

Usando a autenticação básica

Crie um [AmazonCognitoIdentityProviderClient](#) usando [Anonymous AWSCredentials](#), que não exige solicitações assinadas. Você não precisa fornecer uma região. O código subjacente chamará `FallbackRegionFactory.GetRegionEndpoint()` se uma região não for fornecida. Crie objetos `CognitoUserPool` e `CognitoUser`. Chame o método `StartWithSrpAuthAsync` com um `InitiateSrpAuthRequest` que contenha a senha do usuário.

```
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;

}
```

Autenticar com desafios

Continuar o fluxo de autenticação com desafios, como com `NewPasswordRequired` e `Multi-Factor Authentication (MFA)`, também é mais simples. Os únicos requisitos são os `CognitoAuthentication` objetos, a senha do usuário para o SRP e as informações necessárias para o próximo desafio, que são adquiridas após a solicitação do usuário para inseri-la. O código a seguir mostra uma forma de verificar o tipo de desafio e obter as respostas apropriadas para `MFA` e `NewPasswordRequired` desafios durante o fluxo de autenticação.

Faça uma solicitação de autenticação básica como antes, e `await` um `AuthFlowResponse`. Quando a resposta for recebida, percorra o objeto retornado `AuthenticationResult`. Se o tipo de `ChallengeName` for `NEW_PASSWORD_REQUIRED`, chame o método `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.");
}
}
}

```

Use AWS recursos após a autenticação

Depois que um usuário é autenticado usando a `CognitoAuthentication` biblioteca, a próxima etapa é permitir que o usuário acesse os AWS recursos apropriados. Para fazer isso, crie um banco de identidades por meio do console de identidades federadas do Amazon Cognito. Ao especificar

o grupo de usuários do Amazon Cognito que você criou como provedor, usando seu poolID e clientID, você pode permitir que grupo de usuários do Amazon Cognito acesse os recursos AWS conectados à sua conta. Você também pode especificar diferentes funções para permitir que usuários autenticados e não autenticados acessem recursos diferentes. Você pode alterar essas regras no console do IAM, em que você pode adicionar ou remover permissões no campo Action (Ação) da política anexada da função. Em seguida, usando o grupo de identidades, o grupo de usuários e as informações de usuário do Amazon Cognito apropriados, você pode fazer chamadas para diferentes AWS recursos. O exemplo a seguir mostra um usuário autenticado com SRP acessando os diferentes buckets do Amazon S3 permitidos pela função do grupo de identidades associado

```
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);
        }
    }
}
```

```
}
```

Opções adicionais de autenticação

Além do SRP e do MFA `NewPasswordRequired`, `CognitoAuthentication` a biblioteca de extensões oferece um fluxo de autenticação mais fácil para:

- Personalizado – iniciar com uma chamada para `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- `RefreshToken` - Inicie com uma chamada para `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- `RefreshTokenSRP` - Inicie com uma chamada para `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- `AdminNoSRP` - Inicie com uma chamada para `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Chame o método adequado de acordo com o fluxo que você deseja. Em seguida, continue abordando o usuário com desafios à medida que são apresentados nos objetos `AuthFlowResponse` de cada chamada de método. Além disso, chame o método de resposta adequado, como `RespondToSmsMfaAuthAsync` para desafios de MFA e `RespondToCustomAuthAsync` para desafios personalizados.

Como usar os bancos de dados NoSQL Amazon DynamoDB

Note

Os modelos de programação nesses tópicos estão presentes tanto no .NET Framework quanto no .NET (Core), mas as convenções de chamada diferem, sejam síncronas ou assíncronas.

O AWS SDK para .NET suporta o Amazon DynamoDB, que é um serviço rápido de banco de dados NoSQL oferecido pela AWS. O SDK fornece três módulos de programação para comunicar-se com o DynamoDB: o modelo de baixo nível, o modelo de documento e o modelo de persistência de objetos.

As informações a seguir apresentam esses modelos e seus APIs, fornecem exemplos de como e quando usá-los e fornecem links para recursos adicionais de programação do DynamoDB no. AWS SDK para .NET

Modelo de baixo nível

O modelo de programação de baixo nível envolve chamadas diretas para o serviço do DynamoDB. Você acessa esse modelo por meio do [namespace DBv2 Amazon.Dynamo](#).

Dos três modelos, o de baixo nível é o que exige que você escreva mais código. Por exemplo, você deve converter tipos de dados .NET para seus equivalentes no DynamoDB. Contudo, esse modelo fornece acesso à maioria dos recursos.

Os exemplos a seguir mostram como usar o modelo de baixo nível para criar ou modificar uma tabela e inserir itens em uma tabela no DynamoDB.

Criar uma tabela

No exemplo a seguir, você cria uma tabela usando o método `CreateTable` da classe `AmazonDynamoDBClient`. O método `CreateTable` usa uma instância da classe `CreateTableRequest` que contém características como nomes de atributo do item obrigatório, definição de chave primária e a capacidade de transferência. O método `CreateTable` retorna uma instância da 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);
}
```

Verificação de que uma tabela está pronta para ser modificada

Antes que você possa alterar ou modificar uma tabela, esta tem de estar pronta para modificação. O exemplo a seguir mostra como usar o modelo de baixo nível para verificar se uma tabela do DynamoDB está pronta. Neste exemplo, a tabela de destino a ser verificada é mencionada pelo método `DescribeTable` da classe `AmazonDynamoDBClient`. A cada cinco segundos, o código

verifica o valor da propriedade `TableStatus` da tabela. Quando o status é definido como `ACTIVE`, a tabela está pronta para ser modificada.

```
// 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);
```

Introdução um item em uma tabela

No exemplo a seguir, você usa o modelo de baixo nível para inserir dois itens em uma tabela do DynamoDB. Cada item é inserido pelo método `PutItem` da classe `AmazonDynamoDBClient` usando uma instância da classe `PutItemRequest`. Cada uma das duas instâncias da classe `PutItemRequest` pega o nome da tabela em que os itens serão introduzidos, com uma série de valores de atributos de item.

```
// 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);
```

Modelo de documento

O modelo de programação de documento oferece uma maneira mais fácil para trabalhar com dados dentro do DynamoDB. Esse modelo é destinado especificamente a acessar tabelas e itens nas tabelas. Você acessa esse modelo por meio do [DBv2Amazon.Dynamo. DocumentModel](#) namespace.

Em comparação com o modelo de programação de baixo nível, o modelo de documento é mais fácil de codificar em relação aos dados do DynamoDB. Por exemplo, não é necessário converter tantos tipos de dados .NET em seus equivalentes no DynamoDB. Contudo, esse modelo não fornece acesso a tantos recursos como o modelo de programação de baixo nível. Por exemplo, você pode usar esse modelo para criar, recuperar, atualizar e excluir itens nas tabelas. No entanto, para criar as tabelas, é preciso usar o módulo de baixo nível. Em comparação com o modelo de persistência

de objetos, este modelo exige que você grave mais código para armazenar, carregar e consultar objetos .NET.

Para obter mais informações sobre o modelo de programação de documentos do DynamoDB, consulte [.NET: modelo de documento](#) no [Amazon DynamoDB Developer Guide](#).

As seções a seguir fornecem informações sobre como criar uma representação da tabela desejada do DynamoDB e exemplos sobre como usar o modelo de documento para inserir itens em tabelas e obter itens de tabelas.

Criar uma representação da tabela

Para realizar operações de dados usando o modelo de documento, você deve primeiro criar uma instância da classe `Table` que representa uma tabela específica. Há duas maneiras Principais de fazer isso:

LoadTable method

O primeiro mecanismo é usar um dos métodos estáticos `LoadTable` da classe [Table](#), semelhante ao exemplo a seguir:

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

Note

Embora esse mecanismo funcione, em determinadas condições, ele às vezes pode resultar em latência ou bloqueios adicionais devido aos comportamentos de inicialização a frio e de thread-pool. Para obter mais informações sobre esses comportamentos, consulte a postagem do blog [Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET](#).

TableBuilder

Um mecanismo alternativo, a [TableBuilder](#) classe, foi introduzido na [versão 3.7.203 do pacote .Dynamo AWSSDK](#). DBv2 NuGet Esse mecanismo pode abordar os comportamentos mencionados acima removendo certas chamadas de método implícitas, especificamente, o método `DescribeTable`. Esse mecanismo é usado de maneira semelhante ao seguinte exemplo:

```
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();
```

Para obter mais informações sobre esse mecanismo alternativo, consulte novamente a postagem do blog [Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET](#).

Introdução de um item em uma tabela

No exemplo a seguir, um item é introduzido na tabela de respostas por meio do método `PutItemAsync` da classe `Table`. O método `PutItemAsync` leva uma instância da classe `Document`; a classe `Document` é simplesmente uma coleção de atributos inicializados.

```
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);
```

Obtendo um item de uma tabela

No exemplo a seguir, uma resposta é recuperada pelo método `GetItemAsync` da classe `Table`. Para determinar a resposta a ser obtida, o `GetItemAsync` método usa a chave hash-and-range primária da resposta de destino.

```
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"]);
```

O exemplo anterior converte implicitamente valores de tabela em strings para o método `WriteLine`. Você pode fazer conversões explícitas usando os vários métodos de “As[type]” da classe `DynamoDBEntry`. Por exemplo, você pode converter explicitamente o valor para o `Id` a partir do tipo de dado `Primitive` para um GUID por meio do método `AsGuid()`:

```
var guid = reply["Id"].AsGuid();
```

Modelo de persistência de objeto

O modelo de programação de persistência de objetos é especificamente projetado para armazenar, carregar e consultar objetos .NET no DynamoDB. Você acessa esse modelo por meio do [DBv2Amazon.Dynamo. DataModel](#) namespace.

Dos três modelos, o modelo de persistência de objetos é o mais fácil de codificar sempre que estiver armazenando, carregando ou consultando dados do DynamoDB. Por exemplo, você trabalha diretamente com os tipos de dados do DynamoDB. Contudo, este modelo fornece acesso somente a operações que armazenam, carregam e consultam objetos .NET no DynamoDB. Por exemplo, você pode usar esse modelo para criar, recuperar, atualizar e excluir itens nas tabelas. No entanto, é preciso primeiro criar suas tabelas usando o modelo de baixo nível e, em seguida, usar esse modelo para mapear suas classes .NET para as tabelas.

Para obter mais informações sobre o modelo de programação de persistência de objetos do DynamoDB, consulte [.NET: modelo de persistência de objetos](#) no [Guia do desenvolvedor do Amazon DynamoDB](#).

Os exemplos a seguir mostram como definir uma classe .NET que representa um item DynamoDB, usa uma instância da classe .NET para inserir um item no DynamoDB e usa uma instância de uma classe .NET para obter um item de uma tabela.

Definindo uma classe .NET que represente um item em uma tabela

No exemplo a seguir de uma definição de classe, o `DynamoDBTable` atributo especifica o nome da tabela, enquanto os `DynamoDBRangeKey` atributos `DynamoDBHashKey` e modelam a chave hash-and-range primária da tabela. O atributo `DynamoDBGlobalSecondaryIndexHashKey` é definido para que uma consulta de respostas de um autor específico possa ser construída.

```
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; }
}
```

Criando um contexto para o modelo de persistência de objetos

Para usar o modelo de programação de persistência de objetos para o DynamoDB, você deve criar um contexto que forneça uma conexão ao DynamoDB e permita acessar tabelas, e realizar diversas operações e executar consultas.

Contexto básico

O exemplo de código a seguir mostra como criar o contexto mais básico.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Contexto com DisableFetchingTableMetadata propriedade

O exemplo a seguir mostra como você também pode definir a propriedade `DisableFetchingTableMetadata` da classe `DynamoDBContextConfig` para evitar chamadas implícitas ao método `DescribeTable`.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Se a propriedade `DisableFetchingTableMetadata` estiver definida como `false` (o padrão), conforme mostrado no primeiro exemplo, você poderá omitir os atributos que descrevem a estrutura de chave e índice dos itens da tabela da classe `Reply`. Em vez disso, esses atributos serão inferidos por meio de uma chamada implícita ao método `DescribeTable`. Se o `DisableFetchingTableMetadata` for definido como `true`, conforme mostrado no segundo exemplo, métodos do modelo de persistência de objetos, como o `SaveAsync` e o `QueryAsync` depende inteiramente dos atributos definidos na classe `Reply`. Nesse caso, uma chamada para o método `DescribeTable` não ocorre.

Note

Sob certas condições, as chamadas para o método `DescribeTable` às vezes podem levar a latência ou bloqueios adicionais devido aos comportamentos de inicialização a frio e thread-pool. Por esse motivo, às vezes é vantajoso evitar chamadas para esse método. Para obter mais informações sobre esses comportamentos, consulte a postagem do blog [Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET](#).

Usando uma instância da classe .NET para inserir um item em uma tabela

Neste exemplo, um item é inserido pelo método `SaveAsync` da classe `DynamoDBContext`, que recebe uma instância inicializada da classe .NET que representa o item.

```
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"
});
```

Usando uma instância de um objeto .NET para obter itens de uma tabela

Neste exemplo, uma consulta é criada para encontrar todos os registros de “Author1” usando o método `QueryAsync` da classe `DynamoDBContext`. Em seguida, os itens são recuperados por meio do método `GetNextSetAsync` de consulta.

```
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);  
}
```

Informações adicionais sobre o modelo de persistência de objetos

Os exemplos e explicações mostrados acima às vezes incluem uma propriedade da classe `DynamoDBContext` chamada `DisableFetchingTableMetadata`. Essa propriedade, que foi introduzida na [versão 3.7.203 do DBv2 NuGet pacote AWSSDK .Dynamo](#), permite que você evite certas condições que podem causar latência adicional ou bloqueios devido aos comportamentos de inicialização a frio e pool de threads. Para obter mais informações, consulte a postagem de blog [Padrões aprimorados de inicialização do Dynamo DB para o AWS SDK para .NET](#).

A seguir, algumas informações adicionais sobre essa propriedade.

- Essa propriedade pode ser definida globalmente em seu arquivo `app.config` ou `web.config` se estiver usando o .NET Framework.
- Essa propriedade pode ser definida globalmente usando a classe [AWSConfigsDynamoDB](#), como mostra o exemplo a seguir.

```
// Set the DisableFetchingTableMetadata property globally  
// before constructing any context objects.  
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;  
  
var client = new AmazonDynamoDBClient();  
var context = new DynamoDBContext(client);
```

- Em alguns casos, você não pode adicionar atributos do DynamoDB a uma classe .NET, por exemplo, se a classe estiver definida em uma dependência. Nesses casos, ainda é possível aproveitar a propriedade `DisableFetchingTableMetadata`. Para fazer isso, use a classe [TableBuilder](#) além da propriedade `DisableFetchingTableMetadata`. A `TableBuilder` classe também foi introduzida na [versão 3.7.203 do pacote .Dynamo AWSSDK](#). 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"
});
```

Mais informações

Usando o AWS SDK para .NET para programar o DynamoDB, informações e exemplos

- [DynamoDB APIs](#)
- [Kickoff do DynamoDB Series](#)
- [DynamoDB Series – Modelo de documento](#)
- [DynamoDB Series – Esquemas de conversão](#)
- [DynamoDB Series – Modelo de persistência de objetos](#)
- [DynamoDB Series – Expressões](#)
- [Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET](#)
- [Suporte para JSON no Amazon DynamoDB](#)

Informações e exemplos do modelo de baixo nível

- [Trabalhando com tabelas usando a API AWS SDK para .NET de baixo nível](#)
- [Trabalhando com itens usando a API AWS SDK para .NET de baixo nível](#)
- [Consultando tabelas usando a API de AWS SDK para .NET baixo nível](#)
- [Digitalizando tabelas usando a API AWS SDK para .NET de baixo nível](#)
- [Trabalhando com índices secundários locais usando a API AWS SDK para .NET de baixo nível](#)

- [Trabalhando com índices secundários globais usando a API AWS SDK para .NET de baixo nível](#)

Informações e exemplos do modelo de documentos

- [Tipos de dados do DynamoDB](#)
- [Dinamo DBEntry](#)
- [.NET: Modelo de documento](#)

Informações e exemplos de modelo de persistência de objetos

- [.NET: Modelo de persistência de objeto](#)

Outras informações úteis

- Consulte [Integrar a AWS ao .NET Aspire](#) para acessar informações sobre o desenvolvimento com o Amazon DynamoDB local por meio do .NET Aspire.

Tópicos

- [Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET](#)
- [Suporte para JSON no Amazon DynamoDB](#)

Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

Os exemplos de código a seguir demonstram como usar o para AWS SDK para .NET programar o DynamoDB com expressões. As expressões denotam os atributos que você deseja ler de um item na tabela do DynamoDB. Você também usa expressões ao gravar um item, de forma a indicar quaisquer condições devam ser atendidas (também conhecido como atualização condicional) e como os atributos devem ser atualizados. Alguns exemplos de atualização substituem o atributo por um valor novo ou adicionam novos dados a uma lista ou um mapa. Para obter mais informações,

consulte [Reading and Writing Items Using Expressions \(Leitura e gravação de itens usando expressões\)](#).

Tópicos

- [Dados de exemplo](#)
- [Obtenha um único item usando expressões e a chave principal do item](#)
- [Obtenha vários itens usando expressões e a chave principal da tabela](#)
- [Obtenha vários itens usando expressões e outros atributos do item](#)
- [Imprimir um item](#)
- [Criar ou substituir um item usando expressões](#)
- [Atualização de um item usando expressões](#)
- [Exclusão de um item usando expressões](#)
- [Mais informações](#)

Dados de exemplo

Os exemplos de código neste tópico dependem dos dois itens de exemplo a seguir em uma tabela do DynamoDB chamada ProductCatalog. Esses itens descrevem as informações sobre entradas do produto em um catálogo fictício de loja de bicicleta. Esses itens são baseados no exemplo fornecido em [Estudo de caso: um ProductCatalog item](#). Os descritores de tipos de dados, como BOOL, L, M, N, NS, S e SS, corresponde aos em [JSON Data Format \(Formato de dados JSON\)](#).

```
{
  "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."
            ]
        }
    }
}
}
}
}

```

Obtenha um único item usando expressões e a chave principal do item

O exemplo a seguir apresenta o método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` e um conjunto de expressões para obter e imprimir o item que tem um `Id` de 205. Somente os atributos a seguir do item são retornados: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` e `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);
```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o placeholder `#pr` para representar o atributo `ProductReviews` e o placeholder `#ri` para representar o atributo `RelatedItems`. A chamada para `PrintItem` refere-se a uma função personalizada, como descrito em [Imprimir um item](#).

Obtenha vários itens usando expressões e a chave principal da tabela

O exemplo a seguir caracteriza o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` e uma série de expressões para obter e imprimir o item que tenha um `Id` de 301, mas somente se o valor de `Price` for maior que 150. Somente os atributos a seguir do item são retornados: `Id`, `Title` e todos os atributos `ThreeStar` em `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("====");
}

```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o espaço reservado `#pr` para representar o atributo `ProductReviews` e o espaço reservado `#p` para representar o atributo `Price`. `#pr.ThreeStar` especifica para retornar somente o atributo `ThreeStar`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:val` para representar o valor 150. A propriedade `FilterExpression` especifica que `#p` (`Price`) deve ser maior que `:val` (150). A chamada para `PrintItem` refere-se a uma função personalizada, como descrito em [Imprimir um item](#).

Obtenha vários itens usando expressões e outros atributos do item

O exemplo a seguir apresenta o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` e um conjunto de expressões para obter e imprimir todos os itens que tenham `ProductCategory` of `Bike`. Somente os atributos a seguir do item são retornados: `Id`, `Title` e todos os atributos em `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("====");
}

```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o placeholder `#pr` para representar o atributo `ProductReviews` e o placeholder `#pc` para representar o atributo `ProductCategory`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:catg` para representar o valor `Bike`. A propriedade `FilterExpression` especifica que `#pc` (`ProductCategory`) deve ser igual a `:catg` (`Bike`). A chamada para `PrintItem` refere-se a uma função personalizada, como descrito em [Imprimir um item](#).

Imprimir um item

O exemplo a seguir mostra como imprimir os atributos e os valores de um item. Este exemplo é usado nos exemplos anteriores que mostram como [Obter um único item usando expressões e a chave principal do item](#), [Obter vários itens usando expressões e a chave principal da tabela](#) e [Obter vários itens usando expressões e outros atributos do item](#).

```

// 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");
}

```

No exemplo anterior, cada valor de atributo tem várias data-type-specific propriedades que podem ser avaliadas para determinar o formato correto para imprimir o atributo. Essas propriedades incluem B, BOOL, BS, L, M, N, NS, NULL, S e SS, que correspondem às no [formato de dados JSON](#). Para propriedades como B, N, NULL e S, se a propriedade correspondente não for null, o atributo será o tipo de dados não null correspondente. Para propriedades como BS, L, MNS, e SS, se Count for maior que zero, o atributo será do tipo de non-zero-value dados correspondente. Se todas as data-type-specific propriedades do atributo forem iguais null ou Count iguais a zero, o atributo corresponderá ao tipo de BOOL dados.

Criar ou substituir um item usando expressões

O exemplo a seguir apresenta o método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` e um conjunto de expressões para

atualizar o item com Title of 18-Bicycle 301. Se o item não existir ainda, será adicionado um novo item.

```
// 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);
```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:product` para representar o valor `18-Bicycle 301`. A propriedade `ConditionExpression` especifica que `#title` (`Title`) deve ser igual a `:product` (`18-Bicycle 301`). A chamada para `CreateItemData` refere-se à seguinte função personalizada:

```
// 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;
}

```

No exemplo anterior, um item de exemplo com dados de exemplo é retornado ao chamador. Uma série de atributos e valores correspondentes é construída usando os tipos de dados como BOOL, L, M, N, NS, S e SS, que correspondem aos do [JSON Data Format \(formato de dados JSON\)](#).

Atualização de um item usando expressões

O exemplo a seguir apresenta o método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` e um conjunto de expressões para alterar Title para 18" Girl's Bike para o item com Id de 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);

```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:newproduct` para representar o valor `18" Girl's Bike`. A propriedade `UpdateExpression` especifica a alteração de `#title` (`Title`) para `:newproduct` (`18" Girl's Bike`).

Exclusão de um item usando expressões

O exemplo a seguir caracteriza o método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` e um conjunto de expressões para excluir o item com `Id` de `301`, mas somente se o `Title` do item for `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);
```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:product` para representar o valor `18-Bicycle 301`. A propriedade `ConditionExpression` especifica que `#title (Title)` deve ser igual a `:product (18-Bicycle 301)`.

Mais informações

Para obter mais informações e exemplos de código, consulte:

- [DynamoDB Series – Expressões](#)
- [Acesso aos atributos do item com expressões de projeção](#)
- [Uso de placeholders para nomes e valores de atributo](#)
- [Especificação de condições com expressões de condição](#)
- [Modificação de itens e atributos com expressões de atualização](#)
- [Trabalhando com itens usando a API AWS SDK para .NET de baixo nível](#)
- [Consultando tabelas usando a API de AWS SDK para .NET baixo nível](#)
- [Digitalizando tabelas usando a API AWS SDK para .NET de baixo nível](#)
- [Trabalhando com índices secundários locais usando a API AWS SDK para .NET de baixo nível](#)
- [Trabalhando com índices secundários globais usando a API AWS SDK para .NET de baixo nível](#)

Suporte para JSON no Amazon DynamoDB

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

O AWS SDK para .NET suporta dados JSON ao trabalhar com o Amazon DynamoDB. Isso permite que você obtenha dados formatados para JSON com mais facilidade e insira documentos em JSON nas tabelas do DynamoDB.

Tópicos

- [Obtenha dados de uma tabela do DynamoDB em formato JSON](#)
- [Insira os dados do formato JSON em uma tabela do DynamoDB](#)
- [Conversões do tipo de dados do DynamoDB para JSON](#)
- [Mais informações](#)

Obtenha dados de uma tabela do DynamoDB em formato JSON

O exemplo a seguir mostra como obter dados de uma tabela em formato 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  
//    }
```

No exemplo anterior, o método `ToJson` da classe `Document` converte um item da tabela em uma string formatada para JSON. O item é recuperado pelo método `GetItem` da classe `Table`. Para determinar o item a ser obtido, neste exemplo, o `GetItem` método usa a chave hash-and-range primária do item de destino. Para determinar a tabela de onde se obter o item, o método `LoadTable` da classe `Table` usa uma instância da classe `AmazonDynamoDBClient` e o nome da tabela de destino no `DynamoDB`.

Inserira os dados do formato JSON em uma tabela do `DynamoDB`

O exemplo a seguir mostra como usar o formato JSON para inserir um item em uma tabela do `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);
```

No exemplo anterior, o método `FromJson` da classe `Document` converte uma string formatada para JSON em um item. O item é inserido na tabela pelo método `PutItem` da classe `Table`, que usa a instância da classe `Document` que contém o item. Para determinar a tabela na qual inserir o item, o método `LoadTable` da classe `Table` é chamado, especificando uma instância da classe `AmazonDynamoDBClient` e o nome da tabela de destino no `DynamoDB`.

Conversões do tipo de dados do `DynamoDB` para JSON

Sempre que você chamar o método `ToJson` da classe `Document` e depois nos dados JSON resultantes chamar o método `FromJson` para converter dados JSON de volta a uma instância de uma classe `Document`, alguns tipos de dados do `DynamoDB` não serão convertidos como esperado. Especificamente:

- Os conjuntos do `DynamoDB` (os tipos `SS`, `NS` e `BS`) serão convertidos em matrizes JSON.

- Conjuntos e escalares binários do DynamoDB (os tipos B e BS) serão convertidos em strings JSON codificadas para base64 ou em listas de strings.

Nesse cenário, você deverá chamar o método `DecodeBase64Attributes` da classe `Document` para substituir os dados de JSON codificados para base64 pela representação binária correta. O exemplo a seguir substitui um atributo de item escalar binário codificado para base64 em uma instância de uma classe `Document`, de nome `Picture`, pela representação binária correta. Este exemplo também faz o mesmo pelo atributo do item do conjunto binário codificado para base64 na mesma instância da classe `Document`, de nome `RelatedPictures`:

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Mais informações

Para obter mais informações e exemplos de programação de JSON com o DynamoDB com o, consulte: AWS SDK para .NET

- [Suporte a JSON pelo DynamoDB](#)
- [Atualização do Amazon DynamoDB – JSON, nível gratuito expandido, escalabilidade flexível, itens maiores](#)

Trabalhar com o Amazon EC2

O AWS SDK para .NET suporta o [Amazon EC2](#), que é um serviço web que fornece capacidade de computação redimensionável. Você usa essa capacidade de computação para criar e hospedar seus sistemas de software.

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon EC2. As APIs permitem que você trabalhe com recursos do EC2, como grupos de segurança e pares de chaves. As APIs também permitem que você controle as instâncias do Amazon EC2. Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a [Referência da AWS SDK para .NET API](#) (e vá até “Amazon.ec2”).

O Amazon EC2 APIs é fornecido pelo [AWSSDK NuGet pacote.EC2](#).

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Sobre os exemplos

Os exemplos nesta seção mostram como trabalhar com clientes do Amazon EC2 e como gerenciar as instâncias do Amazon EC2.

Os [tutoriais de instâncias spot do EC2](#) mostram como solicitar instâncias spot do Amazon EC2. As instâncias spot permitem que você acesse a capacidade do EC2 não utilizada por um valor mais baixo que o preço sob demanda.

Tópicos

- [Como trabalhar com grupos de segurança no Amazon EC2](#)
- [Trabalhar com pares de chaves do Amazon EC2](#)
- [Consulte regiões e zonas de disponibilidade \(AZs\) no Amazon EC2](#)
- [Trabalhar com as instâncias do Amazon EC2](#)
- [Tutorial de instância spot do Amazon EC2](#)

Como trabalhar com grupos de segurança no Amazon EC2

No Amazon EC2, um grupo de segurança atua como um firewall virtual que controla o tráfego de rede para uma ou mais instâncias do EC2. Por padrão, o EC2 associa as instâncias a um grupo de segurança que não permite tráfego de entrada. Você pode criar um security group que permita que as instâncias do EC2 aceitem um determinado tráfego. Por exemplo, se você precisar se conectar a uma instância do EC2 em Windows, deverá configurar o security group para permitir tráfego RDP.

Para ler mais sobre grupos de segurança, consulte [Grupos de segurança do Amazon EC2](#) no [Guia de usuário do Amazon EC2](#).

Warning

O EC2-Classic foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classic para uma VPC. Para obter mais informações, consulte a postagem do blog [EC2-Classic Networking is Retiring – Here's How to Prepare](#).

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). [Trabalhar com o Amazon EC2](#)

Tópicos

- [Como enumerar grupos de segurança](#)
- [Como criar grupos de segurança](#)
- [Como atualizar Grupos de Segurança](#)

Como enumerar grupos de segurança

Este exemplo mostra como usar o AWS SDK para .NET para enumerar grupos de segurança. Se você fornecer um [Amazon Private Cloud](#) ID, o aplicativo enumera os grupos de segurança dessa VPC específica. Caso contrário, o aplicativo exibirá apenas uma lista de todos os grupos de segurança disponíveis.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Como enumerar seus grupos de segurança](#)
- [Código completo](#)
- [Considerações adicionais](#)

Como enumerar seus grupos de segurança

O snippet a seguir enumera seus grupos de segurança. Ele enumera todos os grupos ou os grupos de uma VPC específica, se houver um.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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();
}
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)
Classe [Amazon EC2 Client](#)
- Namespace [Amazon.EC2.model](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

Classe [Filter](#)

Classe [SecurityGroup](#)

O Código

```
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();
    }
}

```

```
}  
}  
}
```

Considerações adicionais

- Observe que, no caso da VPC, o filtro é construído com a parte Name do par nome-valor definida como “vpc-id”. Esse nome vem da descrição da `Filters` propriedade da [DescribeSecurityGroupsRequest](#) classe.
- Para obter a lista completa dos seus grupos de segurança, você também pode usar [DescribeSecurityGroupsAsync](#) sem parâmetros.
- Você pode verificar os resultados consultando a lista de grupos de segurança no [console do Amazon EC2](#).

Como criar grupos de segurança

Este exemplo mostra como usar o AWS SDK para .NET para criar um grupo de segurança. Você pode fornecer o ID de uma VPC existente para criar um grupo de segurança para o EC2 em uma VPC. Se você não fornecer essa ID, o novo grupo de segurança será para o EC2-Classik se sua AWS conta suportar isso.

Se você não fornecer uma VPC ID e sua AWS conta não for compatível com o EC2-Classik, o novo grupo de segurança pertencerá à VPC padrão da sua conta.

Warning

O EC2-Classik foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classik para uma VPC. Para acessar mais informações, consulte a postagem do blog [EC2-Classik Networking is Retiring - Here's How to Prepare](#).

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Como encontrar grupos de segurança existentes](#)

- [Criar um grupo de segurança](#)
- [Código completo](#)

Como encontrar grupos de segurança existentes

O snippet a seguir pesquisa grupos de segurança existentes com o nome fornecido na VPC fornecida.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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;  
}
```

Criar um grupo de segurança

O snippet a seguir cria um novo grupo de segurança se um grupo com esse nome não existir na VPC fornecida. Se nenhuma VPC for fornecida e existir um ou mais grupos com esse nome, o snippet apenas retornará a lista de grupos.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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 exists, 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;
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [CreateSecurityGroupRequest](#)

Classe [CreateSecurityGroupResponse](#)

Classe [DescribeSecurityGroupsRequest](#)

Classe [DescribeSecurityGroupsResponse](#)

Classe [Filter](#)

Classe [SecurityGroup](#)

O código

```
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(
            $"One or more security groups with name {groupName} already exist.\n");
        return securityGroups;
    }

    // If the security group doesn't already exists, 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);
    }
}
```

Como atualizar Grupos de Segurança

Este exemplo mostra como usar o AWS SDK para .NET para adicionar uma regra a um grupo de segurança. Em particular, o exemplo adiciona uma regra para permitir tráfego de entrada em uma determinada porta TCP, que pode ser usada, por exemplo, para conexões remotas com uma instância do EC2. O aplicativo usa a ID de um grupo de segurança existente, um endereço IP (ou intervalo de endereços) no formato CIDR e, opcionalmente, um número de porta TCP. Em seguida, ele adiciona uma regra de entrada ao grupo de segurança fornecido.

Note

Para usar esse exemplo, você precisa de um endereço IP (ou intervalo de endereços) no formato CIDR. Consulte Considerações adicionais no final deste tópico para obter o endereço IP do seu computador local.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Como adicionar uma regra de entrada](#)
- [Código completo](#)

- [Considerações adicionais](#)

Como adicionar uma regra de entrada

O trecho a seguir adiciona uma regra de entrada a um grupo de segurança para um determinado endereço IP (ou intervalo) e porta TCP.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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($"New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [AuthorizeSecurityGroupIngressRequest](#)

Classe [AuthorizeSecurityGroupIngressResponse](#)

Classe [IpPermission](#)

Classe [IpRange](#)

O código

```
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 eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"New RDP rule was written in {groupId} for {ipAddress}.");
    Console.WriteLine($"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);  
    }  
}  
  
}
```

Considerações adicionais

- Se você não fornecer um número de porta, o aplicativo usará como padrão a porta 3389. Essa é a porta do Windows RDP, que permite que você se conecte a uma instância do EC2 executando o Windows. Se você iniciar uma instância EC2 do Linux, use a porta TCP 22 (SSH).
- Observe que o exemplo é definido como `IpProtocol "tcp"`. Os valores de `IpProtocol` podem ser encontrados na descrição da `IpProtocol` propriedade da [IpPermission](#) classe.
- Talvez você queira o endereço IP do computador local ao usar este exemplo. A seguir estão algumas das maneiras pelas quais você pode obter o endereço.
 - Se seu computador local (do qual você se conectará à instância do EC2) tiver um endereço IP público estático, você poderá usar um serviço para obter esse endereço. Um desses serviços é o <http://checkip.amazonaws.com/>. Para ler mais sobre a autorização de tráfego de entrada, consulte [Regras de grupos de segurança](#) e [Regras de grupo de segurança para diferentes casos de uso](#) no [Guia do usuário do Amazon EC2](#).
 - Outra maneira de obter o endereço IP do seu computador local é usar o [console do Amazon EC2](#).

Selecione um dos seus grupos de segurança, selecione a guia Regras de entrada e escolha Editar regras de entrada. Em uma regra de entrada, abra o menu suspenso na coluna Fonte e escolha Meu IP para ver o endereço IP do seu computador local no formato CIDR. Certifique-se de cancelar a operação.
- Você pode verificar os resultados desse exemplo examinando a lista de grupos de segurança no [console do Amazon EC2](#).

Trabalhar com pares de chaves do Amazon EC2

O Amazon EC2 utiliza criptografia de chave pública para criptografar e descriptografar as informações de login. A criptografia de chave pública usa uma chave pública para criptografar dados

e, em seguida, o destinatário usa a chave privada para descriptografar os dados. As chaves pública e privada são conhecidas como par de chaves. Se quiser fazer login em uma instância do EC2, você deve especificar um par de chaves ao iniciá-la e, em seguida, fornecer a chave privada do par ao se conectar a ela.

Ao iniciar uma instância do EC2, é possível criar um par de chave para ela ou usar um que já foi usado ao iniciar outras instâncias. Para ler mais sobre pares de chaves do Amazon EC2, consulte [Working with Amazon EC2 key pairs](#) no [Guia do usuário do Amazon EC2](#).

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). [Trabalhar com o Amazon EC2](#)

Tópicos

- [Criar e exibir pares de chaves](#)
- [Excluir pares de chaves](#)

Criar e exibir pares de chaves

Este exemplo mostra como usar o AWS SDK para .NET para criar um par de chaves. O aplicativo usa o nome do novo par de chaves e o nome de um arquivo PEM (com a extensão “.pem”). Ele cria o par de chaves, grava a chave privada no arquivo PEM e exibe todos os pares de chaves disponíveis. Se você não fornecer argumentos de linha de comando, o aplicativo apenas exibirá todos os pares de chaves disponíveis.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Criar o par de chaves](#)
- [Exibir pares de chaves disponíveis](#)
- [Código completo](#)
- [Considerações adicionais](#)

Criar o par de chaves

O snippet a seguir cria um par de chaves e, em seguida, armazena a chave privada no arquivo PEM fornecido.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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);  
    }  
}
```

Exibir pares de chaves disponíveis

O snippet a seguir exibe uma lista dos pares de chaves disponíveis.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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}");  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [CreateKeyPairRequest](#)

Classe [CreateKeyPairResponse](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyPairInfo](#)

O código

```
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($"\\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);
    }
}

//
// 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);
}
}
}
```

Considerações adicionais

- Depois de executar o exemplo, você pode ver o novo par de chaves no [console do Amazon EC2](#).
- Ao criar um par de chaves, você deve salvar a chave privada que é retornada, pois não vai poder recuperá-la depois.

Excluir pares de chaves

Este exemplo mostra como usar o AWS SDK para .NET para excluir um par de chaves. O aplicativo usa o nome de um par de chave. Ele exclui o par de chave e, em seguida, exibe todos os pares de chaves disponíveis. Se você não fornecer argumentos de linha de comando, o aplicativo apenas exibirá todos os pares de chaves disponíveis.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Criar pares de chaves](#)
- [Exibir pares de chaves disponíveis](#)
- [Código completo](#)

Criar pares de chaves

O snippet a seguir exclui um par de chave.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// Method to delete a key pair  
private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)  
{  
    await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{  
        KeyName = keyName});  
    Console.WriteLine($"{keyName} has been deleted (if it existed).");  
}
```

Exibir pares de chaves disponíveis

O snippet a seguir exibe uma lista dos pares de chaves disponíveis.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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}");  
}
```


Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [DeleteKeyPairRequest](#)

Classe [DescribeKeyPairsResponse](#)

Classe [KeyPairInfo](#)

O código

```
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}");
}
}
}

```

Consulte regiões e zonas de disponibilidade (AZs) no Amazon EC2

O Amazon EC2 está hospedado em vários locais do mundo. Esses locais são compostos por regiões da e zonas de disponibilidade. Cada região é uma área geográfica separada, com vários locais isolados, conhecidos como zonas de disponibilidade.

Para ler mais sobre regiões e zonas de disponibilidade, consulte [Regiões e zonas de disponibilidade](#) no [Guia do usuário do Amazon EC2](#).

Este exemplo mostra como usar o AWS SDK para .NET para obter detalhes sobre as regiões e zonas de disponibilidade relacionadas a um cliente EC2. O aplicativo exibe listas das regiões e zonas de disponibilidade disponíveis para um cliente EC2.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [DescribeAvailabilityZonesResponse](#)

Classe [DescribeRegionsResponse](#)

Classe [AvailabilityZone](#)

Classe [Region](#)

```
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);
    }
}
```

Trabalhar com as instâncias do Amazon EC2

Você pode usar o AWS SDK para .NET para controlar instâncias do Amazon EC2 com operações como criar, iniciar e encerrar. Os tópicos nesta seção fornecem alguns exemplos de como fazer isso. Para ler mais sobre instâncias do EC2, consulte as [Instâncias do Amazon EC2](#) no [Guia do usuário do Amazon EC2](#).

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). [Trabalhar com o Amazon EC2](#)

Tópicos

- [Iniciar uma instância do Amazon EC2](#)
- [Encerramento de uma instância Amazon EC2](#)

Iniciar uma instância do Amazon EC2

Este exemplo mostra como usar o AWS SDK para .NET para iniciar uma ou mais instâncias do Amazon EC2 configuradas de forma idêntica a partir da mesma Amazon Machine Image (AMI). Ao usar [várias entradas](#) que você fornece, o aplicativo executa uma instância do EC2 e monitora a instância até que ela saia do estado “Pendente”.

Quando a instância do EC2 estiver em execução, será possível se conectar a ela remotamente, conforme descrito em [\(opcional\) Conecte-se à instância](#).

Warning

O EC2-Classic foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classic para uma VPC. Para acessar mais informações, consulte a postagem do blog [EC2-Classic Networking is Retiring - Here's How to Prepare](#).

As seções a seguir fornecem snippets e outras informações para este exemplo. O [código completo do exemplo](#) é mostrado após os snippets e pode ser criado e executado como está.

Tópicos

- [Colete o que você precisa](#)
- [Iniciar uma instância](#)
- [Monitorar instância](#)

- [Código completo](#)
- [Considerações adicionais](#)
- [\(opcional\) Conecte-se à instância](#)
- [Limpeza](#)

Colete o que você precisa

Para iniciar uma instância do EC2, você precisará de várias coisas.

- Uma [VPC](#) onde a instância será iniciada. Se for uma instância do Windows e você estiver se conectando a ela por meio de RDP, a VPC provavelmente precisará de um gateway da Internet anexado a ela, bem como uma entrada para o gateway de Internet na tabela de rotas. Para obter mais informações, consulte [Gateways da Internet](#) no Guia do usuário da Amazon VPC.
- O ID da sub-rede na qual a instância será iniciada. Uma maneira fácil de encontrar ou criar isso é fazer login no [console da Amazon VPC](#), mas você também pode obtê-lo programaticamente usando os métodos e. [CreateSubnetAsyncDescribeSubnetsAsync](#)

 Note

Se você não fornecer esse parâmetro, a nova instância será executada na VPC padrão da conta.

- O ID de um grupo de segurança existente que pertence à VPC em que a instância será iniciada. Para obter mais informações, consulte [Como trabalhar com grupos de segurança no Amazon EC2](#).
- Se você quiser se conectar à nova instância, o grupo de segurança mencionado anteriormente deve ter uma regra de entrada apropriada que permita tráfego SSH na porta 22 (instância Linux) ou tráfego RDP na porta 3389 (instância do Windows). Para obter informações sobre como fazer isso, consulte [Como atualizar Grupos de Segurança](#), inclusive o [Considerações adicionais](#) próximo ao final desse tópico.
- A imagem de máquina da Amazon (AMI) que será usada para criar a instância. Para obter informações sobre AMIs, consulte [Amazon Machine Images \(AMIs\)](#) no Guia do [usuário do Amazon EC2](#). Em particular, consulte [Encontre uma AMI](#) e [compartilhe AMIs](#).

- O nome de um par de chave do EC2 existente, que é usado para se conectar à nova instância. Para obter mais informações, consulte [Trabalhar com pares de chaves do Amazon EC2](#).
- O nome do arquivo PEM que contém a chave privada do par de chave EC2 mencionado anteriormente. O arquivo PEM é usado quando você se [conecta remotamente](#) à instância.

Iniciar uma instância

O seguinte snippet inicia uma instância do EC2.

Os exemplos [no final deste tópico](#) mostram o snippet em uso.

```
//
// 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;
}
```

Monitorar instância

O snippet a seguir monitora a instância até que ela saia do estado “Pendente”.

Os exemplos [no final deste tópico](#) mostram o snippet em uso.

Consulte a [InstanceState](#) classe para ver os valores válidos da `Instance.State.Code` propriedade.

```
//
```

```

// 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}");  
    }  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

Classe [InstanceType](#)

- Namespace [Amazon.EC2.model](#)

Classe [DescribeInstancesRequest](#)

Classe [DescribeInstancesResponse](#)

Classe [Instância](#)

Classe [InstanceNetworkInterfaceSpecification](#)

Classe [RunInstancesRequest](#)

Classe [RunInstancesResponse](#)

O código

```
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.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}");
    }
}

//
// 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);
}
}
}

```

Considerações adicionais

- Ao verificar o estado de uma instância do EC2, você pode adicionar um filtro à `Filter` propriedade do [DescribeInstancesRequest](#) objeto. Usando essa técnica, você pode limitar a solicitação a determinadas instâncias; por exemplo, instâncias com uma tag específica determinada pelo usuário.
- Para resumir, algumas propriedades receberam valores típicos. Em vez disso, qualquer uma, ou todas essas propriedades, podem ser determinadas programaticamente ou por meio da entrada do usuário.
- Os valores que você pode usar para as `MaxCount` propriedades `MinCount` e do [RunInstancesRequest](#) objeto são determinados pela zona de disponibilidade de destino e pelo número máximo de instâncias permitidas para o tipo de instância. Para obter mais informações, consulte [Quantas instâncias posso executar no Amazon EC2?](#) nas perguntas frequentes (FAQs) do Amazon EC2.
- Se você quiser usar um tipo de instância diferente deste exemplo, há alguns tipos para escolher. Para acessar mais informações, consulte [Tipos de instâncias do Amazon EC2](#) no [Guia do usuário do Amazon EC2](#). Consulte também [Detalhes do tipo de instância](#) e [Explorador de tipos de instância](#).
- Você também pode anexar um [perfil do IAM](#) a uma instância ao iniciá-la. Para fazer isso, crie um [IamInstanceProfileSpecification](#) objeto cuja `Name` propriedade esteja definida como o nome de uma função do IAM. Em seguida, adicione esse objeto à `IamInstanceProfile` propriedade do [RunInstancesRequest](#) objeto.

Note

Para iniciar uma instância do EC2 com um perfil do IAM anexado, uma configuração de um usuário do IAM deve incluir determinadas permissões. Para acessar mais informações sobre as permissões necessárias, consulte [Conceder permissão a um usuário para passar um perfil do IAM para uma instância](#) no [Guia do usuário do Amazon EC2](#).

(opcional) Conecte-se à instância

Depois que uma instância estiver sendo executada, você pode conectar-se a ela remotamente usando o cliente remoto apropriado. Para instâncias Linux e Windows, você precisa do endereço IP público ou do nome DNS público da instância. Você precisará também dos seguintes itens:

Para instâncias do Linux

Também é possível usar um cliente SSH para conectar-se à sua instância do Linux. Certifique-se de que o grupo de segurança que você usou ao iniciar a instância permita tráfego SSH na porta 22, conforme descrito em [Como atualizar Grupos de Segurança](#).

Você também precisa da parte privada do par de chaves que você usou para iniciar a instância, ou seja, o arquivo PEM.

Para acessar mais informações, consulte [Conectar-se à instância do Linux](#) no Guia do usuário do Amazon EC2.

Para instâncias do Windows

Também é possível usar um cliente RDP para conectar-se à sua instância. Certifique-se de que o grupo de segurança que você usou ao iniciar a instância permita tráfego RDP na porta 3389, conforme descrito em [Como atualizar Grupos de Segurança](#).

Você precisará também da senha do administrador. Você pode obter isso usando o código de exemplo a seguir, que exige o ID da instância e a parte privada do par de chave usado para iniciar a instância, ou seja, o arquivo PEM.

Para acessar mais informações, consulte [Conectar-se à sua instância do Windows](#) no Guia do usuário do Amazon EC2.

Warning

Esse código de exemplo retorna a senha de administrador em texto simples para sua instância.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [Amazon.EC2.model](#)

Classe [GetPasswordDataRequest](#)

Classe [GetPasswordDataResponse](#)

O código

```
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($"The password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
    }
    return password;
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(

```

```

        "\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);
}
}
}

```

Limpeza

Quando você não precisar mais da instância do EC2, encerre-a, conforme descrito em [Encerramento de uma instância Amazon EC2](#).

Encerramento de uma instância Amazon EC2

Quando uma ou mais instâncias do Amazon EC2 não forem mais necessárias, é possível encerrá-las.

Este exemplo mostra como usar o para AWS SDK para .NET encerrar instâncias do EC2. A entrada requer um ID de instância.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

- Namespace [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);
    }
}
}
}

```

Depois de executar o exemplo, é uma boa ideia fazer login no [console do Amazon EC2](#) para verificar se a [instância do EC2](#) foi encerrada.

Tutorial de instância spot do Amazon EC2

Este tutorial mostra como usar o AWS SDK para .NET para gerenciar instâncias spot do Amazon EC2.

Visão geral do

Instâncias spot permitem solicitar capacidade não utilizada do Amazon EC2 por um valor mais baixo que o preço sob demanda. Isso pode reduzir significativamente os custos do EC2 para aplicativos que podem ser interrompidos.

O resumo a seguir detalha como as instâncias spot são solicitadas e usadas.

1. Criação uma solicitação de instância spot, especificando o preço máximo que está disposto a pagar.
2. Quando a solicitação for atendida, execute a instância como faria com qualquer outra instância do Amazon EC2.
3. Execute a instância por quanto tempo quiser e, em seguida, encerre-a, a menos que o preço de spot mude de maneira que a instância fique encerrada para você.
4. Limpe a solicitação da Instância Spot quando não for mais necessária, para que as Instâncias Spot não sejam mais criadas.

Essa tem sido uma visão geral de alto nível das Instâncias Spot. Para ter uma melhor compreensão das instâncias spot, consulte [Instâncias spot](#) no [Guia do usuário do Amazon EC2](#).

Sobre este tutorial

Ao seguir este tutorial, você usa o AWS SDK para .NET para fazer o seguinte:

- Criar uma solicitação de instância spot
- Determinar quando a solicitação de instância spot é atendida
- Cancelar a solicitação de instância spot
- Encerrar as instâncias associadas

As seções a seguir fornecem snippets e outras informações para este exemplo. O [código completo do exemplo](#) é mostrado após os snippets e pode ser criado e executado como está.

Tópicos

- [Pré-requisitos](#)
- [Colete o que você precisa](#)
- [Como criar uma solicitação de instância spot](#)
- [Determinar o estado da solicitação de instância spot](#)

- [Limpar suas solicitações de instância spot](#)
- [Limpar as instâncias spot](#)
- [Código completo](#)
- [Considerações adicionais](#)

Pré-requisitos

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). [Trabalhar com o Amazon EC2](#)

Colete o que você precisa

Para criar uma solicitação de instância spot, você precisará de várias coisas.

- O número da instância e o seu tipo. Há alguns tipos de instância a escolher. Para obter mais informações, consulte [Tipos de instâncias do Amazon EC2](#) no [Guia do usuário do Amazon EC2](#). Consulte também [Detalhes do tipo de instância](#) e [Explorador de tipos de instância](#).

Para número padrão para este tutorial é o 1.

- A imagem de máquina da Amazon (AMI) que será usada para criar a instância. Para obter informações sobre AMIs, consulte [Amazon Machine Images \(AMIs\)](#) no Guia do [usuário do Amazon EC2](#). Em particular, consulte [Encontre uma AMI](#) e [compartilhe AMIs](#).
- O preço máximo que você está disposto a pagar por hora por instância. Você pode ver os preços de todos os tipos de instâncias (tanto para instâncias sob demanda quanto para instâncias spot) na [página de preços do Amazon EC2](#). O preço padrão deste tutorial será explicado posteriormente.
- Se você quiser se conectar remotamente a uma instância, a um grupo de segurança com a configuração e os recursos adequados. Isso está descrito em [Como trabalhar com grupos de segurança no Amazon EC2](#) e as informações sobre como [coletar o que você precisa](#) e [conectar-se a uma instância](#) em [Iniciar uma instância do Amazon EC2](#). Para simplificar, este tutorial usa o grupo de segurança chamado padrão, que todas as contas AWS mais recentes possuem.

Há muitas maneiras de abordar a solicitação de instâncias spot. Estas são estratégias comuns:

- Faça solicitações que certamente custarão menos que os preços sob demanda.
- Solicitar com base no valor da computação resultante.
- Solicitar para adquirir capacidade de computação o mais rápido possível.

As explicações a seguir se referem ao [histórico de preços de instâncias spot](#) no [Guia do usuário do Amazon EC2](#).

Reduzir custos abaixo sob demanda

Você tem um job de processamento em lote que modera determinado número de horas ou dias para ser executado. Contudo, você tem flexibilidade em relação ao início e ao fim. Você quer ver se pode o concluído por um custo inferior ao das instâncias sob demanda.

Você examina o histórico de preços spot para tipos de instância usando o console da Amazon EC2 ou a API do Amazon EC2. Depois de analisar o histórico de preços do tipo de instância desejado em determinada zona de disponibilidade, há duas abordagens alternativas para sua solicitação:

- Especificar uma solicitação na extremidade superior do intervalo de preços spot que ainda estão abaixo do preço sob demanda, prevendo que sua solicitação instância spot única seria provavelmente cumprida e executada pelo tempo de computação consecutivo suficiente para concluir o trabalho.
- Especificar uma solicitação na extremidade inferior do intervalo de preço e planejar a combinação de várias instâncias executadas ao longo do tempo por meio de uma requisição persistente. As instâncias seriam executadas por tempo suficiente, ao todo, para concluir o trabalho com um custo total ainda inferior.

Não pagar mais que o valor do resultado

Você tem um job de processamento de dados para ser executado. Você entende o valor dos resultados do trabalho bem o suficiente para saber o quanto valem em termos de custos computação.

Após analisar o histórico de preços spot para seu tipo de instância, escolha o preço de lance no qual o custo do tempo computacional não é mais que o valor dos resultados do job. Você cria uma requisição persistente e a deixa ser executada de forma intermitente, à medida que o preço spot flutua acima ou abaixo da sua solicitação.

Adquirir capacidade de computação rapidamente

Você tem uma necessidade imprevista e de curto prazo para capacidade adicional indisponível pelas instâncias sob demanda. Depois de analisar o histórico de preços spot para seu tipo de instância, você escolhe um preço acima do preço histórico mais alto para aumentar significativamente a probabilidade de sua solicitação ser atendida com rapidez e continuar a computação até a conclusão.

Depois de coletar o que precisa e escolher uma estratégia, você está pronto para solicitar uma Instância Spot. Para este tutorial, o preço máximo da instância spot padrão está definido para ser o mesmo que o preço da instância sob demanda (que é 0,003 USD para este tutorial). Definir o preço desta forma maximiza as chances de que o pedido seja atendido.

Como criar uma solicitação de instância spot

O snippet a seguir mostra como criar uma solicitação de instância spot com os elementos coletados anteriormente.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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];  
}
```

O valor importante retornado desse método é o ID da solicitação da Instância Spot, que está contido no `SpotInstanceRequestId` membro do [SpotInstanceRequest](#) objeto retornado.

 Note

Você será cobrado por todas as instâncias spot que forem iniciadas. Para evitar custos desnecessários, certifique-se de [cancelar todas as solicitações](#) e [encerrar todas as instâncias](#).

Determinar o estado da solicitação de instância spot

O snippet a seguir mostra como obter informações sobre sua solicitação de instância spot. Você pode usar essas informações para tomar certas decisões em seu código, como decidir continuar esperando o atendimento de uma solicitação de instância spot.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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];  
}
```

O método retorna informações sobre a solicitação da instância spot, como o ID da instância, seu estado e o código de status. Para acessar mais informações sobre os códigos de status das solicitações de Instância spot, consulte [Spot request status](#) no [Guia do usuário do Amazon EC2](#).

Limpar suas solicitações de instância spot

Quando você não precisar mais solicitar instâncias spot, é importante cancelar todas as solicitações pendentes para evitar que essas solicitações sejam atendidas novamente. O snippet a seguir mostra como cancelar uma solicitação de instância spot.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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);  
}
```

Limpar as instâncias spot

Para evitar custos desnecessários, é importante que você encerre todas as instâncias que foram iniciadas a partir de solicitações de instância spot. Apenas cancelar a solicitação de instância spot não encerrará as instâncias, o que significa que você continuará a ser cobrado por elas. O snippet a seguir mostra como encerrar uma instância depois de obter o identificador da instância para uma instância spot ativa.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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");
        }
    }
}

```

Código completo

O exemplo de código a seguir chama os métodos descritos anteriormente para criar e cancelar uma solicitação de instância spot e encerrar uma instância spot.

Referências do SDK

NuGet pacotes:

- [AWSSDK.EC2](#)

Elementos de programação:

- Namespace [Amazon.EC2](#)

Classe [Amazon EC2 Client](#)

Classe [InstanceType](#)

- Namespace [Amazon.EC2.model](#)

Classe [CancelSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsRequest](#)

Classe [DescribeSpotInstanceRequestsResponse](#)

Classe [InstanceStateChange](#)

Classe [LaunchSpecification](#)

Classe [RequestSpotInstancesRequest](#)

Classe [RequestSpotInstancesResponse](#)

Classe [SpotInstanceRequest](#)

Classe [TerminateInstancesRequest](#)

Classe [TerminateInstancesResponse](#)

O código

```
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");
    }
}
}
```

Considerações adicionais

- Depois de executar o tutorial, é uma boa ideia fazer login no [console do Amazon EC2](#) para verificar se a [solicitação da Instância Spot](#) foi cancelada e se a [Instância Spot](#) foi encerrada.

Acessando AWS Identity and Access Management (IAM) com o AWS SDK para .NET

O AWS SDK para .NET suporta [AWS Identity and Access Management](#), que é um serviço da web que permite AWS aos clientes gerenciar usuários e permissões de usuário em AWS.

Um usuário AWS Identity and Access Management (IAM) é uma entidade que você cria em AWS. A entidade representa uma pessoa ou aplicativo com AWS quem interage. Para obter mais informações sobre os usuários do IAM, consulte [Usuários do IAM](#) e [limites do IAM e do STS](#) no Guia do usuário do IAM.

Conceder permissões a um usuário criando uma política do IAM. A política contém um documento de política que lista as ações que um usuário pode executar e os recursos que essas ações podem afetar. Para obter mais informações sobre as políticas do IAM, consulte [Permissões e políticas](#) no Guia do usuário do IAM.

Warning

Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como [Centro de Identidade do AWS IAM](#).

APIs

Eles AWS SDK para .NET fornecem APIs para clientes do IAM. Eles APIs permitem que você trabalhe com recursos do IAM, como usuários, funções e chaves de acesso.

Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a [Referência da AWS SDK para .NET API](#) (e vá até “Amazon. IdentityManagement”).

Esta seção também contém [um exemplo](#) que mostra como anexar um perfil do IAM às instâncias do Amazon EC2 para facilitar o gerenciamento de credenciais.

O IAM APIs é fornecido pelo [AWSSDK. IdentityManagement](#) NuGetpacote.

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

Tópicos

- [Criar políticas gerenciadas pelo IAM a partir do JSON](#)
- [Exibir o documento da política gerenciada do IAM](#)
- [Conceder acesso utilizando um perfil do IAM](#)

Criar políticas gerenciadas pelo IAM a partir do JSON

Este exemplo mostra como usar o AWS SDK para .NET para criar uma [política gerenciada do IAM](#) a partir de um determinado documento de política em JSON. O aplicativo cria um objeto cliente do IAM, lê o documento de política de um arquivo e, em seguida, cria a política.

Note

Para ver um exemplo de documento de política em JSON, consulte as [considerações adicionais](#) no final deste tópico.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Crie a política do](#)
- [Código completo](#)
- [Considerações adicionais](#)

Crie a política do

O trecho a seguir cria uma política gerenciada pelo IAM com o nome e o documento de política fornecidos.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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)});  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.IdentityManagement](#)

Elementos de programação:

- [Namespace Amazon. IdentityManagement](#)
Classe [AmazonIdentityManagementServiceClient](#)
- [Namespace Amazon. IdentityManagement.Modelo](#)

Classe [CreatePolicyRequest](#)

Classe [CreatePolicyResponse](#)

O código

```
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);
}
}
}
}

```

Considerações adicionais

- Veja a seguir um exemplo de documento de política que você pode copiar em um arquivo JSON e usar como entrada para esse aplicativo:

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": "*"
}
]
```

- Você pode verificar se a política foi criada consultando o [console do IAM](#). Usar a lista suspensa Filtrar políticas para selecionar Gerenciado pelo cliente. Excluir uma política quando não precisar mais dela.
- Para obter mais informações sobre a criação de políticas, consulte [Criação de políticas do IAM](#) e a [referência da política JSON](#) no [Guia do usuário do IAM](#)

Exibir o documento da política gerenciada do IAM

Este exemplo mostra como usar o AWS SDK para .NET para exibir um documento de política. O aplicativo cria um objeto cliente do IAM, encontra a versão padrão da política gerenciada do IAM em questão e, em seguida, exibe o documento da política em formato JSON.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Encontrar a versão padrão](#)
- [Exibir o documento da política](#)
- [Código completo](#)

Encontrar a versão padrão

Os snippet a seguir encontram a versão padrão da política do IAM.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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;  
}
```

Exibir o documento da política

O snippet a seguir exibe o documento de política JSON da política do IAM fornecida.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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)}}");  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.IdentityManagement](#)

Elementos de programação:

- [Namespace Amazon. IdentityManagement](#)
Classe [AmazonIdentityManagementServiceClient](#)
- [Namespace Amazon. IdentityManagement.Modelo](#)
Classe [GetPolicyVersionRequest](#)
Classe [GetPolicyVersionResponse](#)
Classe [ListPolicyVersionsRequest](#)
Classe [ListPolicyVersionsResponse](#)
Classe [PolicyVersion](#)

O código

```
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)}");
}
}
}

```

Conceder acesso utilizando um perfil do IAM

Este tutorial mostra como usar o para AWS SDK para .NET habilitar funções do IAM em instâncias do Amazon EC2.

Visão geral do

Todas as solicitações AWS devem ser assinadas criptograficamente usando credenciais emitidas por. AWS Portanto, é necessária uma estratégia para gerenciar credenciais para aplicações executadas nas instâncias do Amazon EC2. É necessário distribuir, armazenar e girar essas credenciais com segurança, mas também mantendo-as acessíveis à aplicação.

Com os perfis do IAM, você pode gerenciar essas credenciais com eficiência. Você cria um perfil do IAM e a configura com as permissões que uma aplicação exige e, em seguida, anexa esse perfil a uma instância do EC2. Para ler mais sobre os benefícios de usar perfis do IAM, consulte [Perfis do IAM para o Amazon EC2](#) no [Guia do usuário do Amazon EC2](#). Consulte também as informações sobre [perfis do IAM](#) no Guia do usuário do IAM.

Para um aplicativo criado usando o AWS SDK para .NET, quando o aplicativo constrói um objeto cliente para um AWS serviço, o objeto pesquisa credenciais de várias fontes potenciais. A ordem da pesquisa é mostrada em [Resolução de perfil e credenciais](#).

Se o objeto cliente não encontrar as credenciais em nenhuma outra fonte, ele recuperará as credenciais temporárias com as mesmas permissões que as associadas ao perfil do IAM pelos metadados da instância EC2. Essas credenciais são usadas para fazer chamadas a AWS partir do objeto cliente.

Sobre este tutorial

Ao seguir este tutorial, você usa a AWS SDK para .NET (e outras ferramentas) para iniciar uma instância do Amazon EC2 com uma função do IAM anexada e, em seguida, vê um aplicativo na instância usando as permissões da função do IAM.

Tópicos

- [Criar uma amostra da aplicação no Amazon S3](#)
- [Criar um perfil do IAM](#)
- [Iniciar uma instância do EC2 e anexar o perfil do IAM](#)
- [Conectar-se à instância EC2](#)
- [Executar o exemplo da aplicação na instância do EC2](#)
- [Limpeza](#)

Criar uma amostra da aplicação no Amazon S3

Esta aplicação de exemplo recupera um objeto do Amazon S3. Para executar a aplicação, você precisa:

- Um bucket do Amazon S3 que contém um arquivo de texto.
- AWS credenciais em sua máquina de desenvolvimento que permitem acessar o bucket.

Para ter informações sobre como criar um bucket do Amazon S3 e fazer upload de um objeto, consulte o [Guia do usuário do Amazon Simple Storage Service](#). Para obter informações sobre AWS credenciais, consulte [Configure a autenticação do SDK com AWS](#).

Criar um projeto .NET Core com o seguinte código. Em seguida, teste aplicação em sua máquina de desenvolvimento.

Note

Em sua máquina de desenvolvimento, o domínio .NET Core Runtime está instalado, o que permite que você execute a aplicação sem publicá-la. Ao criar uma instância do EC2 mais adiante neste tutorial, é possível instalar o .NET Core Runtime na instância. Isso proporciona uma experiência semelhante e uma transferência de arquivos menor.

No entanto, é possível optar por não instalar o .NET Core Runtime na instância. Se você escolher esse curso de ação, deverá publicar a aplicação para que todas as dependências sejam incluídas ao transferi-la para a instância.

Referências do SDK

NuGet pacotes:

- [AWSSDK.S3](#)

Elementos de programação:

- Namespace [Amazon.S3](#)
 - Classe [AmazonS3Client](#)
- Namespace [Amazon.S3.Model](#)

Classe [GetObjectResponse](#)

O código

```
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);
}
}
```

Se quiser, você pode remover temporariamente as credenciais usadas na máquina de desenvolvimento para ver como a aplicação responde. (Mas não se esqueça de restaurar as credenciais quando concluir).

Criar um perfil do IAM

Criar um perfil do IAM com as permissões apropriadas para acessar o Amazon S3.

1. Abra o [console do IAM](#).
2. No painel de navegação, escolha Perfis e depois escolha Criar perfil.
3. Selecionar serviço da AWS , escolher EC2 e depois Avançar: Permissões.
4. Em Anexar políticas de permissões, encontre e selecione ReadOnlyAccessAmazonS3. Revise a política, se quiser, e escolha Avançar: Tags.
5. Adicione tags, se quiser, e escolha Avançar: Revisão.

6. Digite um nome e uma descrição para a função e, em seguida, escolha Criar função. Lembre-se desse nome, pois você precisará dele quando executar sua instância do EC2.

Iniciar uma instância do EC2 e anexar o perfil do IAM

Iniciar uma instância do EC2 com o perfil do IAM que foi criada. Você pode fazer isso das seguintes maneiras.

- Usar o console do EC2

Para executar uma instância usando o console do EC2, consulte [Iniciar uma instância usando o novo assistente de inicialização de instâncias](#) no [Guia do usuário do Amazon EC2](#).

Ao examinar a página de lançamento, você deve pelo menos expandir o painel Detalhes avançados para poder especificar o perfil do IAM que você criou anteriormente no perfil da instância do IAM.

- Usando o AWS SDK para .NET

Para obter informações sobre isso, consulte [Iniciar uma instância do Amazon EC2](#), incluindo o [Considerações adicionais](#) próximo ao final desse tópico.

Para iniciar uma instância do EC2 com um perfil do IAM anexado, uma configuração de um usuário do IAM deve incluir determinadas permissões. Para acessar mais informações sobre as permissões necessárias, consulte [Conceder permissão a um usuário para passar um perfil do IAM para uma instância](#) no [Guia do usuário do Amazon EC2](#).

Conectar-se à instância EC2

Conectar à instância do EC2 para que você possa transferir a amostra do aplicativo e, em seguida, executar o aplicativo. Você precisará do arquivo que contém a parte privada do par de chaves usado para executar a instância, ou seja, o arquivo PEM.

Para acessar informações sobre como se conectar à instância, consulte [Conectar-se à sua instância do Linux](#) ou [Conectar-se à instância do Windows](#) no [Guia do usuário do Amazon EC2](#). Ao conectar-se, faça isso de maneira que possa transferir arquivos da sua máquina de desenvolvimento para sua instância.

Se estiver usando o Visual Studio no Windows, também poderá conectar-se à instância usando o Toolkit for Visual Studio. Para obter mais informações, consulte [Conectando-se a uma instância do Amazon EC2](#) no Guia do AWS Toolkit for Visual Studio usuário.

Executar o exemplo da aplicação na instância do EC2

1. Copiar as aplicações do aplicativo da sua unidade local para sua instância.

Os arquivos transferidos dependem de como a aplicação foi criada e se sua instância tem o .NET Core Runtime instalado. Para acessar informações sobre como transferir arquivos para sua instância, consulte [Conecte-se à sua instância Linux](#) (consulte a subseção apropriada) ou [Transferir arquivos para instâncias do Windows](#) no [Guia do usuário do Amazon EC2](#).

2. Iniciar a aplicação e verificar se é executada com os mesmos resultados da sua máquina de desenvolvimento.
3. Verifique se a aplicação usa as credenciais fornecidas pelo perfil do IAM.
 - a. Abra o [console do Amazon EC2](#).
 - b. Selecione a instância e desanexe o perfil do IAM por meio de Ações, Configurações da instância e Anexar/substituir perfil do IAM.
 - c. Execute o aplicativo novamente e veja se retorna um erro de autorização.

Limpeza

Ao terminar este tutorial, se não desejar mais a instância do EC2 criada, certifique-se de encerrar a instância para evitar custos indesejados. Isso poderá ser feito no [console do Amazon EC2](#) ou programaticamente, conforme descrito em [Encerramento de uma instância Amazon EC2](#). Se desejar, você também pode excluir outros recursos que tenha criado para este tutorial. Isso pode incluir um perfil do IAM, um par de chaves do EC2 e um arquivo PEM, um grupo de segurança etc.

Usar o armazenamento para internet Amazon Simple Storage Service

O AWS SDK para .NET suporta o [Amazon S3](#), que é armazenamento para a Internet. Ele foi projetado para facilitar a computação de escala na web para os desenvolvedores.

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon S3. Eles APIs permitem que você trabalhe com recursos do Amazon S3, como buckets e itens. Para ver o conjunto completo de APIs para o Amazon S3, veja o seguinte:

- [AWS SDK para .NET Referência da API](#) (e vá até “Amazon.S3”).
- Documentação [Amazon.Extensions.S3.Encryption](#)

O Amazon S3 APIs é fornecido pelos seguintes pacotes: NuGet

- [AWSSDK.S3](#)
- [Amazon.Extensions.S3.Encryption](#)

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Exemplos neste documento

Os tópicos a seguir neste documento mostram como usar o AWS SDK para .NET para trabalhar com o Amazon S3.

- [Usar chaves KMS para a criptografia do S3](#)

Exemplos em outros documentos

Os links a seguir para o [Amazon S3 Developer Guide](#) fornecem exemplos adicionais de como usar o AWS SDK para .NET para trabalhar com o Amazon S3.

Note

Embora esses exemplos e considerações adicionais de programação tenham sido criados para a versão 3 do AWS SDK para .NET uso do .NET Framework, eles também são viáveis para versões posteriores do AWS SDK para .NET uso do .NET Core. Às vezes, pequenos ajustes no código são necessários.

Exemplos de programação do Amazon S3

- [Gerenciando ACLs](#)
- [Criação de um bucket](#)

- [Upload de um objeto](#)
- [Upload em várias partes com a API de alto nível \(Amazon.S3.Transfer\). TransferUtility](#)
- [Multipart Upload com a API de baixo nível](#)
- [Listar objetos](#)
- [Chaves de listagem](#)
- [Obter um objeto](#)
- [Copiar um objeto](#)
- [Copiar um objeto com a API do Multipart Upload](#)
- [Exclusão de um objeto](#)
- [Excluir de vários objetos](#)
- [Restaurar um objeto](#)
- [Configurar um bucket para notificações](#)
- [Gerenciar o ciclo de vida de um objeto](#)
- [Gerenciar um URL de objeto pré-assinado](#)
- [Gerenciamento de sites](#)
- [Ativação do compartilhamento de recursos de origem cruzada \(CORS\)](#)

Considerações adicionais sobre programação

- [Usar o AWS SDK para .NET para o Amazon S3 Programming](#)
- [Fazer solicitações usando as credenciais temporárias do usuário do IAM](#)
- [Fazer solicitações usando as credenciais temporárias de usuário federado](#)
- [Especificar a criptografia no lado do servidor](#)
- [Especificar a criptografia do lado do servidor com chaves de criptografia fornecidas pelo cliente](#)

Usando AWS KMS chaves para criptografia do Amazon S3 no AWS SDK para .NET

Este exemplo mostra como usar AWS Key Management Service chaves para criptografar objetos do Amazon S3. O aplicativo cria uma chave mestra do cliente (CMK) e a usa para criar um objeto [AmazonS3 EncryptionClient V2](#) para criptografia do lado do cliente. O aplicativo usa esse cliente para criar um objeto criptografado a partir de um determinado arquivo de texto em um bucket existente do Amazon S3. Em seguida, ele descriptografa o objeto e exibe seu conteúdo.

⚠ Warning

Uma classe similar chamada `AmazonS3EncryptionClient` está obsoleta e é menos segura do que a classe `AmazonS3EncryptionClientV2`. Para migrar o código existente que usa `AmazonS3EncryptionClient`, consulte [Migração do cliente de criptografia S3 \(V1 para V2\)](#).

Tópicos

- [Criar materiais de criptografia](#)
- [Criar e criptografar um objeto do Amazon S3](#)
- [Código completo](#)
- [Considerações adicionais](#)

Criar materiais de criptografia

O trecho a seguir cria um objeto `EncryptionMaterials` que contém uma ID de chave KMS.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
// 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);
```

Criar e criptografar um objeto do Amazon S3

O snippet a seguir cria um objeto `AmazonS3EncryptionClientV2` que usa os materiais de criptografia criados anteriormente. Em seguida, ele usa o cliente para criar e criptografar um novo objeto do Amazon S3.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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
    });
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [Amazon.Extensions.S3.Encryption](#)

Elementos de programação:

- Namespace [Amazon.Extensions.S3.Encryption](#)

Classe [EncryptionClientAmazonS3](#) V2

Classe [CryptoConfigurationAmazonS3](#) V2

Classe [CryptoStorageMode](#)

Classe [EncryptionMaterialsV2](#)

- Namespace [Amazon.Extensions.S3.Encryption.Primitives](#)

Classe [KmsType](#)

- Namespace [Amazon.S3.Model](#)

Classe [GetObjectRequest](#)

Classe [GetObjectResponse](#)

Classe [PutObjectRequest](#)

- [Namespace Amazon. KeyManagementService](#)

Classe [AmazonKeyManagementServiceClient](#)

- [Namespace Amazon. KeyManagementService.Modelo](#)

Classe [CreateKeyRequest](#)

Classe [CreateKeyResponse](#)

O código

```
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);
}
}
}
}

```

Considerações adicionais

- Você pode verificar os resultados desse exemplo. Para fazer isso, acesse o [console do Amazon S3](#) e abra o bucket que você forneceu à aplicação. Em seguida, encontre o novo objeto, baixe-o e abra-o em um editor de texto.
- A classe [AmazonS3 EncryptionClient V2](#) implementa a mesma interface da classe padrão. `AmazonS3Client` Isso facilita a portabilidade do código para a classe `AmazonS3EncryptionClientV2` de modo que a criptografia e a descriptografia ocorram automaticamente e de maneira transparente no cliente.
- Uma vantagem de usar uma AWS KMS chave como chave mestra é que você não precisa armazenar e gerenciar suas próprias chaves mestras; isso é feito por AWS. Uma segunda vantagem é que a `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET é interoperável com a `AmazonS3EncryptionClientV2` classe do AWS SDK para Java. Isso significa que você pode criptografar com o AWS SDK para Java e descriptografar com o AWS SDK para .NET e vice-versa.

Note

A `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET suporta chaves mestras do KMS somente quando executada no modo de metadados. O modo do arquivo de instruções da `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET é incompatível com a `AmazonS3EncryptionClientV2` classe do AWS SDK para Java.

- Para obter mais informações sobre a criptografia do lado do cliente com a `AmazonS3EncryptionClientV2` classe e como a criptografia de envelope funciona, consulte Criptografia de [dados do lado do cliente com o Amazon AWS SDK para .NET S3](#).

Enviar notificações da nuvem usando o Amazon Simple Notification Service

Note

As informações neste tópico são específicas para projetos baseados no .NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

O **AWS SDK para .NET** é compatível com o **Amazon Simple Notification Service (Amazon SNS)**, que é um serviço web que permite que aplicativos, usuários finais e dispositivos enviem notificações instantaneamente da nuvem. Para obter mais informações, consulte [Amazon SNS](#).

Listar seus tópicos do Amazon SNS

O exemplo a seguir mostra como listar os tópicos do Amazon SNS, as assinaturas de cada tópico e os atributos de cada tópico. Este exemplo usa o padrão [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));

```

Enviar uma mensagem para um tópico do Amazon SNS

O exemplo a seguir mostra como enviar uma mensagem para um tópico do SNS. O exemplo usa um argumento, o ARN do tópico do 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);
    }
}
}
}

```

Veja o [exemplo completo](#), incluindo informações sobre como criar e executar o exemplo na linha de comando, em GitHub.

Enviar uma mensagem SMS para um número de telefone

O exemplo a seguir mostra como enviar uma mensagem SMS para um número de telefone. O exemplo usa um argumento, o número de telefone, que deve estar em um dos dois formatos descritos nos comentários.

```
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 +1nnnnnnnnnnn
            string number = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
Amazon.RegionEndpoint.USWest2);
            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);
            }
        }
    }
}
```

Veja o [exemplo completo](#), incluindo informações sobre como criar e executar o exemplo na linha de comando, em GitHub.

Enviar mensagens usando o Amazon SQS

O AWS SDK para .NET suporta o [Amazon Simple Queue Service \(Amazon SQS\)](#), que é um serviço de enfileiramento de mensagens que manipula mensagens ou fluxos de trabalho entre componentes em um sistema.

As filas do Amazon SQS fornecem um mecanismo que permite enviar, armazenar e receber mensagens entre componentes de software, como microsserviços, sistemas distribuídos e aplicações sem servidor. Isso permite desacoplar esses componentes e você fica da necessidade de projetar e operar seu próprio sistema de mensagens. Para ter informações sobre como as filas e mensagens funcionam no Amazon SQS, consulte os [tutoriais do Amazon SQS](#) e [Arquitetura básica do Amazon SQS](#) no [Guia do desenvolvedor do Amazon Simple Queue Service](#).

Important

Devido à natureza distribuída da fila, o Amazon SQS não garante que as mensagens serão recebidas na ordem exata de envio. Se for necessário manter a ordem de mensagens, use uma [fila FIFO do Amazon SQS](#).

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon SQS. Eles APIs permitem que você trabalhe com recursos do Amazon SQS, como filas e mensagens. Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a [Referência da AWS SDK para .NET API](#) (e vá até “Amazon.sqs”).

O Amazon SQS é fornecido pelo APIs [AWSSDK NuGet pacote.SQS](#).

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

Tópicos

- [Criação de fila no Amazon SQS](#)
- [Atualizando filas do Amazon SQS](#)
- [Excluindo filas do Amazon SQS](#)
- [Enviando mensagens SMS do Amazon](#)
- [Recebimento de mensagens do Amazon SQS](#)

Criação de fila no Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para criar uma fila do Amazon SQS. O aplicativo cria uma [fila de mensagens não entregues](#) se você não fornecer o ARN para uma. Em seguida, ele cria uma fila de mensagens padrão, que inclui uma fila de mensagens não entregues (a que você forneceu ou a que foi criada).

Se você não fornecer nenhum argumento de linha de comando, a aplicação simplesmente mostrará informações sobre todas as filas existentes.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Mostrar filas existentes](#)
- [Criação da fila](#)
- [Obtenha o ARN de uma fila](#)
- [Código completo](#)
- [Considerações adicionais](#)

Mostrar filas existentes

O trecho a seguir mostra uma lista das filas existentes na região do cliente SQS e os atributos de cada fila.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
```

```
// 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($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"{att.Key}: {att.Value}");
}
```

Criação da fila

O seguinte snippet cria uma fila. O trecho inclui o uso de uma fila de mensagens não entregues, mas uma fila de mensagens não entregues não é necessariamente necessária para suas filas.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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;
}

```

Obtenha o ARN de uma fila

O snippet a seguir obtém o ARN da fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```

//
// 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;
}

```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.SQS](#)

Elementos de programação:

- Namespace [Amazon.SQS](#)

Classe [Amazon SQSClient](#)

Classe [QueueAttributeName](#)

- Namespace [Amazon.SQS.Model](#)

Classe [CreateQueueRequest](#)

Classe [CreateQueueResponse](#)

Classe [GetQueueAttributesResponse](#)

Classe [ListQueuesResponse](#)

O código

```
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);  
    }  
}  
  
}
```

Considerações adicionais

- O nome da fila deve ser composto de caracteres alfanuméricos, hifens e sublinhados.
- Os nomes das filas e as filas diferenciam maiúsculas de URLs minúsculas
- Se precisar do URL da fila, mas tiver apenas o nome da fila, use um dos `AmazonSQSClient.GetQueueUrlAsync` métodos.
- Para obter informações sobre os vários atributos de fila que você pode definir, consulte [CreateQueueRequest](#) na Referência da [AWS SDK para .NET API](#) ou [SetQueueAttributes](#) na Referência da [API do Amazon Simple Queue Service](#).
- Este exemplo especifica uma sondagem longa para todas as mensagens na fila que você cria. Isto é feito usando o atributo `ReceiveMessageWaitTimeSeconds`.

Você também pode especificar uma sondagem longa durante uma chamada para os `ReceiveMessageAsync` métodos da `SQSClient` classe [Amazon](#). Para obter mais informações, consulte [Recebimento de mensagens do Amazon SQS](#).

Para obter informações sobre sondagem curta versus sondagem longa, consulte [Sondagem curta e longa](#) no Guia do desenvolvedor do Amazon Simple Queue Service.

- Uma fila de mensagens não entregues é aquela fila que outras filas (de origem) podem direcionar para mensagens que não foram processadas com sucesso. Para obter mais informações, consulte [filas de mensagens não entregues do Amazon SQS](#) no Guia do desenvolvedor do Amazon Simple Queue Service.
- Você também pode ver a lista de filas e os resultados desse exemplo no [console do Amazon SQS](#).

Atualizando filas do Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para atualizar uma fila do Amazon SQS. Depois de algumas verificações, a aplicação atualiza o atributo fornecido com o valor fornecido e, em seguida, mostra todos os atributos da fila.

Se somente o URL da fila estiver incluído nos argumentos da linha de comando, a aplicação simplesmente mostrará todos os atributos da fila.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Mostrar atributos da fila](#)
- [Validar nome do atributo](#)
- [Atualizar atributo de fila](#)
- [Código completo](#)
- [Considerações adicionais](#)

Mostrar atributos da fila

O snippet a seguir mostra os atributos da fila identificados pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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}");  
}
```

Validar nome do atributo

O snippet a seguir valida o nome do atributo que está sendo atualizado.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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;  
}
```

Atualizar atributo de fila

O snippet a seguir atualiza um atributo da fila identificado pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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}});  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.SQS](#)

Elementos de programação:

- Namespace [Amazon.SQS](#)

Classe [Amazon SQSClient](#)

Classe [QueueAttributeName](#)

- Namespace [Amazon.SQS.Model](#)

Classe [GetQueueAttributesResponse](#)

O código

```
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($"The 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($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"{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);
    }
}
}
}

```

Considerações adicionais

- Para atualizar o `RedrivePolicy` atributo, você deve cotar o valor inteiro e escapar das aspas dos key/value pares, conforme apropriado para seu sistema operacional.

No Windows, por exemplo, o valor é construído de forma semelhante ao seguinte:

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
```

Excluindo filas do Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para excluir uma fila do Amazon SQS. A explicação exclui a fila, espera até um determinado período de tempo até que a fila seja excluída e, em seguida, mostra uma lista de filas restantes.

Se você não fornecer nenhum argumento de linha de comando, a aplicação simplesmente mostrará uma lista de filas existentes.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Excluir a fila](#)
- [Aguardar até a fila ser removida](#)
- [Mostrar uma lista de filas existentes](#)
- [Código completo](#)
- [Considerações adicionais](#)

Excluir a fila

O snippet a seguir exclui a fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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.");  
}
```

Aguardar até a fila ser removida

O snippet a seguir aguarda a conclusão do processo de exclusão, o que pode levar até 60 segundos.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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;
    }
}
```

Mostrar uma lista de filas existentes

O trecho a seguir mostra uma lista das filas existentes na região do cliente SQS.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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}");
}
```



```
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.SQS](#)

Elementos de programação:

- Namespace [Amazon.SQS](#)
Classe [Amazon SQSClient](#)
- Namespace [Amazon.SQS.Model](#)
Classe [ListQueuesResponse](#)

O código

```
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}");
}
}
}

```

Considerações adicionais

- A chamada de API `DeleteQueueAsync` não verifica se a fila que você está excluindo está sendo usada como uma fila de mensagens não entregues. Um procedimento mais sofisticado poderia verificar isso.
- Você também pode ver a lista de filas e os resultados desse exemplo no [console do Amazon SQS](#).

Enviando mensagens SMS do Amazon

[Este exemplo mostra como usar o para enviar mensagens AWS SDK para .NET para uma fila do Amazon SQS, que você pode criar programaticamente ou usando o console do Amazon SQS](#). A aplicação envia uma única mensagem para a fila e, em seguida, um lote de mensagens. Em seguida, a aplicação aguarda a entrada do usuário, que pode ser mensagens adicionais para enviar para a fila ou uma solicitação para sair da aplicação.

Este exemplo e o [próximo exemplo sobre o recebimento de mensagens](#) podem ser usados juntos para ver o fluxo de mensagens no Amazon SQS.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Enviar uma mensagem](#)
- [Enviar um lote de mensagens](#)
- [Excluir todas as mensagens da fila](#)
- [Código completo](#)
- [Considerações adicionais](#)

Enviar uma mensagem

O snippet a seguir envia uma mensagem para a fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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}");
}
```

Enviar um lote de mensagens

O snippet a seguir envia um lote de mensagens para a fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// 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.");
}
```

Excluir todas as mensagens da fila

O snippet a seguir exclui todas as mensagens da fila identificada pelo URL da fila em questão. Isso também é conhecido como limpando a fila.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($"Purging messages from queue\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
}
```

```
Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.SQS](#)

Elementos de programação:

- Namespace [Amazon.SQS](#)

Classe [Amazon SQSClient](#)

- Namespace [Amazon.SQS.Model](#)

Classe [PurgeQueueResponse](#)

Classe [SendMessageBatchResponse](#)

Classe [SendMessageResponse](#)

Classe [SendMessageBatchRequestEntry](#)

Classe [SendMessageBatchResultEntry](#)

O código

```
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 = "{\n\"product\": [\n\"name\": \"Product A\", \"price\
\n\": \"32\"],\n\"name\": \"Product B\", \"price\": \"27\"]}";
    private const string XmlMessage = "<products><product name=\"Product A\" price=
\n\"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($"Purging messages from queue\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
}
}
```

Considerações adicionais

- Para obter informações sobre várias limitações nas mensagens, incluindo os caracteres permitidos, consulte [Cotas relacionadas às mensagens](#) no Guia do [desenvolvedor do Amazon Simple Queue Service](#).
- As mensagens permanecem nas filas até serem excluídas ou a fila ser removida. Quando uma mensagem for recebida por uma aplicação, ela não ficará visível na fila, mesmo que ainda exista na fila. Para obter mais informações sobre tempos limite de visibilidade, consulte [Tempo limite de visibilidade do Amazon SQS](#).
- Além do corpo da mensagem, você também pode adicionar atributos às mensagens. Para obter mais informações, consulte [Mensagem de metadados](#).

Recebimento de mensagens do Amazon SQS

[Este exemplo mostra como usar o AWS SDK para .NET para receber mensagens de uma fila do Amazon SQS, que você pode criar programaticamente ou usando o console do Amazon SQS. A](#)

aplicação lê uma única mensagem da fila, processa a mensagem (nesse caso, exibe o corpo da mensagem no console) e, em seguida, exclui a mensagem da fila. A aplicação repete essas etapas até que o usuário digite uma tecla no teclado.

Este exemplo e o [exemplo anterior sobre envio de mensagens](#), podem ser usados juntos para ver o fluxo de mensagens no Amazon SQS.

As seções a seguir fornecem snippets desse exemplo. O [código completo do exemplo](#) é mostrado depois e pode ser criado e executado como está.

Tópicos

- [Recebimento de uma mensagem](#)
- [Excluir uma mensagem](#)
- [Código completo](#)
- [Considerações adicionais](#)

Recebimento de uma mensagem

O trecho a seguir recebe uma mensagem da fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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.)  
    });  
}
```

Excluir uma mensagem

O snippet a seguir exclui uma mensagem da fila identificada pelo URL da fila em questão.

O exemplo [no final deste tópico](#) mostra o snippet em uso.

```
//  
// 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);  
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- [AWSSDK.SQS](#)

Elementos de programação:

- Namespace [Amazon.SQS](#)

Classe [Amazon SQSClient](#)

- Namespace [Amazon.SQS.Model](#)

Classe [ReceiveMessageRequest](#)

Classe [ReceiveMessageResponse](#)

O código

```
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);
}
}
}

```

Considerações adicionais

- Para especificar uma sondagem longa, este exemplo usa a propriedade `WaitTimeSeconds` de cada chamada ao método `ReceiveMessageAsync`.

Você também pode especificar uma sondagem longa para todas as mensagens em uma fila usando o atributo `ReceiveMessageWaitTimeSeconds` ao [criar](#) ou [atualizar](#) a fila.

Para obter informações sobre sondagem curta versus sondagem longa, consulte [Sondagem curta e longa](#) no Guia do desenvolvedor do Amazon Simple Queue Service.

- Durante o processamento da mensagem, você pode usar o identificador de recebimento para alterar o tempo limite de visibilidade da mensagem. Para obter informações sobre como fazer isso, consulte os `ChangeMessageVisibilityAsync` métodos da `SQSClient` classe [Amazon](#).
- Chamar o método `DeleteMessageAsync` remove a mensagem da fila incondicionalmente, qualquer que seja a configuração do tempo limite de visibilidade.

Usando AWS Lambda para serviço de computação

O AWS SDK para .NET é compatível com AWS Lambda, o que permite que você execute código sem provisionar ou gerenciar servidores. Para obter mais informações, consulte a [página do produto AWS Lambda](#) e o [Guia do desenvolvedor do AWS Lambda](#), especialmente a seção [Trabalhando com C#](#).

APIs

O AWS SDK para .NET fornece APIs para o AWS Lambda. As APIs permitem que você trabalhe com atributos do Lambda, como [funções](#), [gatilhos](#) e [eventos](#). Para ver o conjunto completo de APIs, consulte [Lambda](#) na AWS SDK para .NET [Referência de API do](#) .

As APIs do Lambda são fornecidas pelos [pacotes NuGet](#).

Pré-requisitos

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Mais informações

Consulte [Integrar a AWS ao .NET Aspire](#) para acessar informações sobre o desenvolvimento com o AWS Lambda por meio do .NET Aspire.

Tópicos

Tópicos

- [Usando anotações para escrever funções do AWS Lambda](#)

Usando anotações para escrever funções do AWS Lambda

Ao escrever funções do Lambda, às vezes é necessário escrever uma grande quantidade de código de manipulador e atualizar modelos do AWS CloudFormation, entre outras tarefas. Lambda Annotations é uma estrutura para ajudar a aliviar essas cargas para funções do Lambda do .NET 6, tornando assim a experiência de escrever Lambda em C# mais natural.

Como exemplo do benefício de usar a estrutura do Lambda Annotations, considere os seguintes trechos de código que adicionam dois números.

Sem Lambda Annotations

```
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" } }
        };
    }
}
```

```
}
```

Com Lambda Annotations

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Conforme mostrado no exemplo, o Lambda Annotations pode eliminar a necessidade de um determinado código clichê.

Para obter detalhes sobre como usar a estrutura, bem como informações adicionais, consulte os seguintes recursos:

- O [README do GitHub](#) para documentação sobre as APIs e os atributos do Lambda Annotations.
- A [publicação no blog](#) do Lambda Annotations.
- O pacote NuGet [Amazon.Lambda.Annotations](#).
- O [projeto Photo Asset Management](#) no GitHub. Especificamente, consulte a pasta [PamApiAnnotations](#) e as referências a Lambda Annotations no [README](#) do projeto.

Bibliotecas e estruturas de alto nível para o AWS SDK para .NET

As seções a seguir contêm informações sobre bibliotecas e estruturas de alto nível que não fazem parte da funcionalidade principal do SDK. Essas bibliotecas e estruturas usam a funcionalidade principal do SDK para criar recursos que facilitam determinadas tarefas.

Se você é novo no AWS SDK para .NET, talvez queira conferir o tópico [Faça um tour rápido](#) primeiro. Ele fornece uma introdução ao SDK.

Antes de começar, assegure-se de [configurar o ambiente e o projeto](#). Revise também as informações em [Atributos do SDK](#).

Tópicos

- [Estrutura de processamento de mensagens para .NET da AWS](#)
- [Integrar a AWS ao .NET Aspire no AWS SDK para .NET](#)

Estrutura de processamento de mensagens para .NET da AWS

A Estrutura de processamento de mensagens para .NET da AWS é uma estrutura nativa da AWS que simplifica o desenvolvimento de aplicações de processamento de mensagens .NET que usam serviços da AWS como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon EventBridge. A estrutura reduz a quantidade de código padronizado que os desenvolvedores precisam escrever, permitindo que você se concentre na lógica empresarial ao publicar e consumir mensagens. Para acessar detalhes sobre como o framework pode simplificar seu desenvolvimento, consulte a postagem do blog [Introducing the AWS Message Processing Framework for .NET \(Preview\)](#). A primeira parte, especificamente, oferece uma demonstração que exhibe a diferença entre o uso de chamadas de API de baixo nível e o uso do framework.

O Message Processing Framework comporta as seguintes atividades e recursos:

- Enviar mensagens ao SQS e publicar eventos no SNS e no EventBridge.
- Receber e processar mensagens do SQS usando um agente de sondagem de execução prolongada que, normalmente, é usado em serviços em segundo plano. Isso inclui gerenciar o tempo limite de visibilidade enquanto uma mensagem está sendo processada para evitar que outros clientes a processem.
- Processar mensagens em funções do AWS Lambda.
- Filas FIFO (primeiro a entrar, primeiro a sair) do SQS e tópicos do SNS.
- OpenTelemetry para registro em log.

Para acessar detalhes sobre essas atividades e recursos, consulte a seção Features da [postagem do blog](#) e os tópicos listados abaixo.

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Recursos adicionais

- O pacote [AWS.Messaging](#) em [NuGet.org](#).
- A [referência de API](#).

- O arquivo README no repositório do GitHub em <https://github.com/aws/aws-dotnet-messaging/>.
- [Injeção de dependência do .NET](#) da Microsoft.
- [Host genérico do .NET](#) da Microsoft.

Tópicos

- [Comece a usar o AWS Message Processing Framework para .NET](#)
- [Publicar mensagens com o AWS Message Processing Framework para .NET](#)
- [Consumir mensagens com o AWS Message Processing Framework para .NET](#)
- [Usar FIFO com o AWS Message Processing Framework para .NET](#)
- [Registro em log e Open Telemetry para o AWS Message Processing Framework para .NET](#)
- [Personalizar o AWS Message Processing Framework para .NET](#)
- [Segurança para o AWS Message Processing Framework for .NET](#)

Comece a usar o AWS Message Processing Framework para .NET

Antes de começar, assegure-se de ter [configurado o seu ambiente e seu projeto](#). Revise também as informações em [Atributos do SDK](#).

Este tópico fornece informações que ajudarão você a começar a usar o Message Processing Framework. Além das informações de pré-requisitos e configuração, é fornecido um tutorial que mostra como implementar um cenário comum.

Pré-requisitos e configuração

- As credenciais fornecidas à sua aplicação devem ter as permissões apropriadas para o serviço de mensagens e as operações que ele usa. Para acessar mais informações, consulte os tópicos de segurança para [SQS](#), [SNS](#) e [EventBridge](#) em seus respectivos guias do desenvolvedor. Veja também a parte do arquivo [README](#) no GitHub que aborda [permissões](#) específicas.
- Para usar o AWS Message Processing Framework para .NET, você deve adicionar o pacote NuGet [AWS.Messaging](#) ao seu projeto. Por exemplo:

```
dotnet add package AWS.Messaging
```

- O framework se integra ao [contêiner de serviço de injeção de dependência \(DI\)](#) do .NET. É possível configurar o framework durante a inicialização da sua aplicação chamando `AddAWSMessageBus` para adicioná-lo ao contêiner de 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");
});
```

Tutorial

Este tutorial demonstra como usar o AWS Message Processing Framework para .NET. Ele cria duas aplicações: uma API ASP.NET Core Minimal que envia mensagens a uma fila do Amazon SQS quando recebe uma solicitação em um endpoint da API e uma aplicação de console de execução prolongada que pesquisa essas mensagens e as processa.

- As instruções neste tutorial prestigiam a CLI do .NET, mas você pode seguir esse tutorial usando ferramentas multiplataforma, como a CLI do .NET ou o Microsoft Visual Studio. Para acessar informações sobre ferramentas, consulte [Instale e configure seu conjunto de ferramentas](#).
- Este tutorial pressupõe que você esteja usando o perfil [default] para receber as credenciais. Também pressupõe que credenciais de curto prazo estejam disponíveis com as permissões apropriadas para enviar e receber mensagens do Amazon SQS. Para acessar mais informações, consulte [Configure a autenticação do SDK com AWS](#) e os tópicos sobre segurança para o [SQS](#).

Note

Na execução deste tutorial, é possível que haja custos para o envio de mensagens do SQS.

Etapas

- [Criar uma fila do SQS](#)
- [Criar e executar a aplicação de publicação](#)
- [Criar e executar a aplicação de processamento](#)
- [Limpeza](#)

Criar uma fila do SQS

Este tutorial requer uma fila do SQS para enviar e receber mensagens. É possível criar uma fila com um dos seguintes comandos para a AWS CLI ou o Ferramentas da AWS para PowerShell. Anote o URL da fila exibida para que você possa especificá-la na configuração do framework a seguir.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Ferramentas da AWS para PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Criar e executar a aplicação de publicação

Use o procedimento a seguir para criar e executar a aplicação de publicação.

1. Abra um prompt de comando ou um terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.
2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

```
dotnet new webapi --name Publisher
```

3. Navegue para a pasta do novo projeto. Adicione uma dependência no AWS Message Processing Framework para .NET.

```
cd Publisher  
dotnet add package AWS.Messaging
```

Note

Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar `AWSSDK.SSO` e `AWSSDK.SSO0IDC`.

4. Substitua o código em `Program.cs` pelo código a seguir.

```
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. Execute o comando a seguir. Isso deve abrir uma janela do navegador com o Swagger UI, que permite explorar e testar sua API.

```
dotnet watch run <queue URL created earlier>
```

6. Abra o endpoint /greeting e escolha Testar.
7. Especifique os valores senderName e greeting da mensagem e escolha Executar. Isso invoca sua API, que envia a mensagem do SQS.

Criar e executar a aplicação de processamento

Use o procedimento a seguir para criar e executar a aplicação de processamento.

1. Abra um prompt de comando ou um terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.
2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

```
dotnet new console --name Handler
```

3. Navegue para a pasta do novo projeto. Adicione uma dependência no AWS Message Processing Framework para .NET. Adicione também o pacote Microsoft.Extensions.Hosting, que permite configurar o framework por meio do [host genérico do .NET](#).

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

Note

Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar AWSSDK.SSO e AWSSDK.SSO0IDC.

4. Substitua o código em Program.cs pelo código a seguir.

```
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. Execute o comando a seguir. Isso inicia uma sondagem de execução prolongada.

```
dotnet run <queue URL created earlier>
```

Logo após a inicialização, a aplicação receberá a mensagem que foi enviada na primeira parte deste tutorial e registrará em log a seguinte mensagem:

```
Received message {greeting} from {senderName}
```

6. Pressione Ctrl+C para interromper o agente de sondagem.

Limpeza

Use um dos comandos a seguir para a AWS CLI ou o Ferramentas da AWS para PowerShell para excluir a fila.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Ferramentas da AWS para PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publicar mensagens com o AWS Message Processing Framework para .NET

O AWS Message Processing Framework para .NET comporta a publicação de um ou mais tipos de mensagens, o processamento de um ou mais tipos de mensagens ou a realização de ambos na mesma aplicação.

O código a seguir mostra a configuração de uma aplicação que está publicando diferentes tipos de mensagens em serviços distintos da AWS.

```
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");
});
```

Depois de registrar o framework durante a inicialização, injete o `IMessagePublisher` genérico em seu código. Chame o respectivo método `PublishAsync` para publicar qualquer um dos tipos de mensagem configurados acima. O publicador genérico determinará o destino para o qual direcionar a mensagem com base no tipo.

No exemplo a seguir, um controlador ASP.NET MVC recebe mensagens `ChatMessage` e eventos `OrderInfo` dos usuários e, depois, os publica no Amazon SQS e no Amazon SNS, respectivamente. Os dois tipos de mensagem podem ser publicados usando o publicador genérico configurado acima.

```
[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();
}
}
```

Para direcionar uma mensagem para a lógica de processamento apropriada, o framework usa metadados chamados identificador do tipo de mensagem. Por padrão, esse é o nome completo do

tipo .NET da mensagem, incluindo seu nome de montagem. Se você estiver enviando e processando mensagens, esse mecanismo funcionará bem se compartilhar a definição dos objetos de mensagem entre projetos. No entanto, se as mensagens forem redefinidas em namespaces diferentes ou se você estiver trocando mensagens com outros frameworks ou linguagens de programação, talvez seja necessário substituir o identificador do tipo de mensagem.

```
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");
    });
});
```

Publicadores específicos do serviço

O exemplo mostrado acima usa o `IMessagePublisher` genérico, que pode publicar em qualquer serviço aceito da AWS com base no tipo de mensagem configurado. O framework também fornece publicadores específicos de serviços para Amazon SQS, Amazon SNS e Amazon EventBridge. Esses publicadores específicos expõem opções que se aplicam somente a esse serviço e podem ser injetadas usando os tipos `ISQSPublisher`, `ISNSPublisher` e `IEventBridgePublisher`.

Por exemplo, ao enviar mensagens a uma fila FIFO do SQS, você deve precisa o [ID do grupo de mensagens](#) apropriado. O código a seguir mostra o exemplo `ChatMessage` novamente, mas agora usando um `ISQSPublisher` para definir opções específicas do 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();
}
}

```

O mesmo pode ser feito para o SNS e o EventBridge, utilizando o `ISNSPublisher` e o `IEventBridgePublisher` respectivamente.

```

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>  
});
```

Por padrão, as mensagens de um tipo específico são enviadas ao destino configurado com antecedência. No entanto, é possível substituir o destino de uma única mensagem usando os respectivos publicadores específicos. Você também pode substituir o cliente do AWS SDK para .NET subjacente utilizado para publicar a mensagem, o que pode ser útil em aplicações multilocatárias nas quais você precisa alterar perfis ou credenciais, dependendo do destino.

```
await _sqsPublisher.SendAsync(message, new SQSOptions  
{  
    OverrideClient = <override IAmazonSQS client>,  
    QueueUrl = <override queue URL>  
});
```

Consumir mensagens com o AWS Message Processing Framework para .NET

O AWS Message Processing Framework para .NET possibilita que você consuma mensagens que foram [publicadas](#) utilizando o framework ou um dos serviços de mensagens. As mensagens podem ser consumidas de várias maneiras, algumas das quais estão descritas abaixo.

Manipuladores de mensagens

Para consumir mensagens, implemente um manipulador usando a interface `IMessageHandler` para cada tipo de mensagem que você deseja processar. O mapeamento entre tipos e manipuladores é configurado na inicialização do projeto.

```
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();

```

O código a seguir mostra um exemplo de manipulador para uma mensagem ChatMessage.

```

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());
    }
}

```

O MessageEnvelope externo contém metadados usados pelo framework. A respectiva propriedade message é o tipo de mensagem (nesse caso, ChatMessage).

É possível exibir MessageProcessStatus.Success() para indicar que a mensagem foi processada com êxito e que o framework a excluirá da fila do Amazon SQS. Ao exibir

`MessageProcessStatus.Failed()`, a mensagem permanecerá na fila, onde poderá ser processada novamente ou transferida para uma [fila de mensagens não entregues](#), se configurada.

Tratar mensagens em um processo de longa duração

Você pode chamar `AddSQSPoller` com um URL de fila do SQS para iniciar um [BackgroundService](#) execução prolongada que pesquisará continuamente a fila e processará mensagens.

```
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();
```

Configurar o SQS Message Poller

É possível configurar o SQS Message Poller pelo `SQSMessagePollerOptions` durante a chamada `AddSQSPoller`.

- `MaxNumberOfConcurrentMessages`: o número máximo de mensagens da fila a serem processadas simultaneamente. O valor padrão é 10.
- `WaitTimeSeconds`: a duração (em segundos) pela qual a chamada `ReceiveMessage` do SQS espera que uma mensagem chegue na fila antes de retornar. Se uma mensagem estiver disponível, a chamada retornará antes de `WaitTimeSeconds`. O valor padrão é 20.

Processamento do tempo limite de visibilidade de mensagens

As mensagens do SQS têm um [tempo limite de visibilidade](#). Quando um consumidor começa a processar uma mensagem específica, ela permanece na fila, mas fica oculta dos outros consumidores para que ela não seja processada mais de uma vez. Se a mensagem não for processada e excluída antes de se tornar visível novamente, outro consumidor poderá tentar processar a mesma mensagem.

O framework vai monitorar e tentar estender o tempo limite de visibilidade das mensagens que está processando no momento. É possível configurar esse comportamento em `SQSMessagePollerOptions` ao chamar `AddSQSPoller`.

- `VisibilityTimeout`: a duração em segundos em que as mensagens recebidas permanecem ocultas de solicitações de recuperação subsequentes. O valor padrão é 30.
- `VisibilityTimeoutExtensionThreshold`: quando o tempo limite de visibilidade de uma mensagem estiver a esse intervalo de segundos de expirar, o framework estenderá o tempo limite de visibilidade (por mais `VisibilityTimeout` segundos). O valor padrão é 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`: com que frequência, em segundos, o framework vai conferir as mensagens que estão a esse `VisibilityTimeoutExtensionThreshold` de expirar e, depois, estenderá o tempo limite de visibilidade. O valor padrão é 1.

No exemplo a seguir, o framework vai conferir a cada segundo as mensagens que ainda estão sendo processadas. Para essas mensagens que estão a 5 segundos de se tornarem visíveis novamente, o framework estenderá automaticamente o tempo limite de visibilidade de cada mensagem em mais 30 segundos.

```
// 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;
});
```

Processar mensagens em funções do AWS Lambda

É possível usar o AWS Message Processing Framework para .NET com a [integração do SQS ao Lambda](#). O recurso é fornecido pelo pacote `AWS.Messaging.Lambda`. Consulte o arquivo [README](#) para começar.

Usar FIFO com o AWS Message Processing Framework para .NET

Para casos de uso em que a ordenação e a deduplicação de mensagens são essenciais, o AWS Message Processing Framework para .NET comporta [filas FIFO \(primeiro a sair, primeiro a entrar\) do Amazon SQS](#) e [tópicos do Amazon SNS](#).

Publicação

Ao publicar mensagens em uma fila ou tópico FIFO, você precisa definir o ID do grupo de mensagens, que especifica o grupo ao qual a mensagem pertence. As mensagens em um grupo são processadas em ordem. Você pode definir isso nos publicadores de mensagens específicos do SQS e do SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Assinatura

Ao processar mensagens de uma fila FIFO, o framework lida com elas dentro de um grupo específico na ordem em que foram recebidas para cada chamada `ReceiveMessages`. O framework entra nesse modo de operação automaticamente quando configurado com uma fila que termine com `.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();
```

Registro em log e Open Telemetry para o AWS Message Processing Framework para .NET

O AWS Message Processing Framework para .NET é instrumentada para que o OpenTelemetry registre em log [rastreamentos](#) de cada mensagem publicada ou processada pelo framework. Esse recurso é fornecido pelo pacote [AWS.Messaging.Telemetry.OpenTelemetry](#). Consulte o arquivo [README](#) para começar.

Note

Para acessar informações sobre segurança relacionadas ao registro em log, consulte [Segurança para o AWS Message Processing Framework for .NET](#).

Personalizar o AWS Message Processing Framework para .NET

O AWS Message Processing Framework para .NET cria, envia e processa mensagens em três “camadas” diferentes:

1. Na camada mais externa, o framework cria a solicitação ou resposta nativa da AWS específica de um serviço. Com o Amazon SQS, por exemplo, ele cria solicitações [SendMessage](#) e trabalha com os objetos [Message](#) definidos pelo serviço.
2. Dentro da solicitação e resposta do SQS, o framework define o elemento `MessageBody` (ou `Message` para o Amazon SNS ou `Detail` para o Amazon EventBridge) como um [CloudEvent formatado em JSON](#). Ele contém metadados definidos pelo framework que podem ser acessados no objeto `MessageEnvelope` ao processar uma mensagem.
3. Na camada mais interna, o atributo `data` dentro do objeto JSON do `CloudEvent` contém uma serialização JSON do objeto .NET que foi enviado ou recebido como mensagem.

```
{
  "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>"
}
```

É possível personalizar a forma como o envelope da mensagem é configurado e lido:

- "id" identifica a mensagem de forma exclusiva. Por padrão, ele é definido como um novo GUID, mas isso pode ser substituído implementando o seu próprio `IMessageIdGenerator` e injetando-o no contêiner de DI.
- "type" controla como a mensagem é direcionada para os manipuladores. Por padrão, ele usa o nome completo do tipo .NET que corresponde à mensagem. Você pode substituí-lo por meio do parâmetro `messageTypeIdentifier` ao associar o tipo de mensagem ao destino via `AddSQSPublisher`, `AddSNSPublisher` ou `AddEventBridgePublisher`.
- "source" indica qual sistema ou servidor enviou a mensagem.
 - Esse será o nome da função se estiver publicando do AWS Lambda, o nome do cluster e o ARN da tarefa se estiver no Amazon ECS e o ID da instância se estiver no Amazon EC2, caso contrário, um valor alternativo de `/aws/messaging`.
 - Você pode substituí-lo por meio de `AddMessageSource` ou `AddMessageSourceSuffix` no `MessageBusBuilder`.
- "time" definido como o `DateTime` atual em UTC. Isso pode ser substituído implementando o seu próprio `IDateTimeHandler` e injetando-o no contêiner de DI.
- "data" contém uma representação JSON do objeto .NET que foi enviado ou recebido como mensagem:
 - `ConfigureSerializationOptions` no `MessageBusBuilder` permite que você configure o [System.Text.Json.JsonSerializerOptions](#) que será usado ao serializar e desserializar a mensagem.
 - Para injetar atributos adicionais ou transformar o envelope da mensagem depois que a estrutura o cria, você pode implementar `ISerializationCallback` e registrar isso por meio de `AddSerializationCallback` no `MessageBusBuilder`.

Segurança para o AWS Message Processing Framework for .NET

O AWS Message Processing Framework para .NET depende do AWS SDK para .NET para comunicação com a AWS. Para acessar mais informações sobre segurança no AWS SDK para .NET, consulte [Segurança para este AWS produto ou serviço](#).

Por motivos de segurança, o framework não registra em log mensagens de dados enviadas pelo usuário. Se quiser habilitar essa funcionalidade para fins de depuração, você precisará chamar `EnableDataMessageLogging()` no barramento de mensagens da seguinte forma:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Se você descobrir um possível problema de segurança, consulte a [política de segurança](#) para fornecer informações.

Integrar a AWS ao .NET Aspire no AWS SDK para .NET

O .NET Aspire é uma nova forma de criar aplicações prontas para a nuvem. Especificamente, ele oferece orquestração para ambientes locais nos quais executar, conectar e depurar os componentes de aplicações distribuídas. Para melhorar o ciclo interno de desenvolvimento de aplicações prontas para a nuvem, foram criadas integrações ao .NET Aspire para conectar suas aplicações .NET aos recursos da AWS. Essas integrações estão disponíveis por meio do pacote NuGet [Aspire.Hosting.AWS](#).

As seguintes integrações do .NET Aspire estão disponíveis:

- A capacidade de provisionar seus recursos da AWS por meio do [CloudFormation](#). Essa integração é utilizada no projeto .NET Aspire AppHost.

Para acessar mais informações, consulte a postagem do blog: [Integrating AWS with .NET Aspire](#).

- Instalar, configurar e conectar o AWS SDK para .NET ao [Amazon DynamoDB local](#). Essa integração é utilizada no projeto .NET Aspire AppHost.

Para acessar mais informações, consulte a postagem do blog: [Integrating AWS with .NET Aspire](#).

- Habilite um ambiente de desenvolvimento local para funções do [AWS Lambda](#). Essa integração é utilizada no projeto .NET Aspire AppHost.

Para acessar mais informações, consulte as postagens do blog [Building and Debugging .NET Lambda applications with .NET Aspire \(Part 1\)](#) e [Building and Debugging .NET Lambda applications with .NET Aspire \(Part 2\)](#).

 Note

Esta é uma documentação de pré-lançamento de um recurso em versão de pré-visualização. Está sujeita a alteração.

Como esse recurso está em versão prévia, opte por usar os recursos de visão prévia. Para acessar informações adicionais sobre esse recurso de versão prévia e como aceitá-lo, consulte o [problema do rastreador de desenvolvimento no GitHub](#).

Mais informações

Para acessar informações e detalhes adicionais sobre como você pode usar as integrações disponíveis no [Aspire.Hosting.AWS](#), consulte os recursos a seguir.

- Postagem do blog [Integrating AWS with .NET Aspire](#).
- Postagens do blog [Building and Debugging .NET Lambda applications with .NET Aspire \(Part 1\)](#) e [Building and Debugging .NET Lambda applications with .NET Aspire \(Part 2\)](#).
- O repositório [integrations-on-dotnet-aspire-for-aws](#) no GitHub.
- O arquivo [README](#) detalhado do pacote NuGet [Aspire.Hosting.AWS](#).

Suporte para outros serviços da AWS e configuração

O AWS SDK para .NET oferece suporte aos serviços da AWS, além dos descritos nas seções anteriores. Para obter informações sobre as APIs de todos os serviços compatíveis, consulte a [Referência de API do AWS SDK para .NET](#).

Além dos namespaces para serviços individuais da AWS, o AWS SDK para .NET também fornece as seguintes APIs:

Área	Descrição	Recursos
AWSSuporte ao	Acesso programático a casos do AWS Support e atributos do Trusted Advisor.	Consulte Amazon.AWSSupport e Amazon.AWSSupport.Model .
Geral	Classes auxiliares e enumerações.	Consulte Amazon e Amazon.Util .

SDK para .NET exemplos de código

Os exemplos de código neste tópico mostram como usar o AWS SDK para .NET with AWS.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Alguns serviços contêm categorias de exemplo adicionais que mostram como utilizar bibliotecas ou funções específicas do serviço.

Services

- [Exemplos de ACM usando SDK para .NET](#)
- [Exemplos de API Gateway usando SDK para .NET](#)
- [Exemplos de Aurora usando SDK para .NET](#)
- [Exemplos de Auto Scaling usando SDK para .NET](#)
- [Exemplos do Amazon Bedrock Runtime usando SDK para .NET](#)
- [CloudWatch Exemplos de registros usando SDK para .NET](#)
- [Exemplos de provedores de identidade do Amazon Cognito usando SDK para .NET](#)
- [Exemplos do Amazon Comprehend usando SDK para .NET](#)
- [Exemplos do Amazon DocumentDB usando SDK para .NET](#)
- [Exemplos do DynamoDB usando SDK para .NET](#)
- [EC2 Exemplos da Amazon usando SDK para .NET](#)
- [Exemplos do Amazon ECS usando SDK para .NET](#)
- [Elastic Load Balancing - Exemplos da versão 2 usando SDK para .NET](#)
- [EventBridge exemplos usando SDK para .NET](#)
- [EventBridge Exemplos de agendador usando SDK para .NET](#)

- [Exemplos do Amazon Glacier usando SDK para .NET](#)
- [AWS Glue exemplos usando SDK para .NET](#)
- [Exemplos de IAM usando SDK para .NET](#)
- [Exemplos do Amazon Keyspaces usando SDK para .NET](#)
- [Exemplos de Kinesis usando SDK para .NET](#)
- [AWS KMS exemplos usando SDK para .NET](#)
- [Exemplos de Lambda usando SDK para .NET](#)
- [MediaConvert exemplos usando SDK para .NET](#)
- [Exemplos do Amazon MSK usando SDK para .NET](#)
- [Exemplos de organizações usando SDK para .NET](#)
- [Exemplos do Partner Central usando SDK para .NET](#)
- [Exemplos do Amazon Pinpoint usando SDK para .NET](#)
- [Exemplos do Amazon Polly usando SDK para .NET](#)
- [Exemplos do Amazon RDS usando SDK para .NET](#)
- [Exemplos de serviços de dados do Amazon RDS usando SDK para .NET](#)
- [Exemplos do Amazon Rekognition usando SDK para .NET](#)
- [Exemplos de registro de domínio do Route 53 usando SDK para .NET](#)
- [Exemplos do Amazon S3 usando SDK para .NET](#)
- [SageMaker Exemplos de IA usando SDK para .NET](#)
- [Exemplos de Secrets Manager usando SDK para .NET](#)
- [Exemplos do Amazon SES usando SDK para .NET](#)
- [Exemplos da API v2 do Amazon SES usando SDK para .NET](#)
- [Exemplos do Amazon SNS usando SDK para .NET](#)
- [Exemplos do Amazon SQS usando SDK para .NET](#)
- [Exemplos de Step Functions usando SDK para .NET](#)
- [AWS STS exemplos usando SDK para .NET](#)
- [Suporte exemplos usando SDK para .NET](#)
- [Exemplos do Amazon Textract usando SDK para .NET](#)
- [Exemplos do Amazon Transcribe usando SDK para .NET](#)
- [Exemplos do Amazon Translate usando SDK para .NET](#)

Exemplos de ACM usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o ACM.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

DescribeCertificate

O código de exemplo a seguir mostra como usar DescribeCertificate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}
}

```

- Para obter detalhes da API, consulte [DescribeCertificatea](#) Referência AWS SDK para .NET da API.

ListCertificates

O código de exemplo a seguir mostra como usar ListCertificates.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}

```

- Para obter detalhes da API, consulte [ListCertificates](#) a Referência AWS SDK para .NET da API.

Exemplos de API Gateway usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with API Gateway.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

AWS as contribuições da comunidade são exemplos que foram criados e mantidos por várias equipes AWS. Para deixar seu feedback, use o mecanismo fornecido nos repositórios vinculados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Cenários](#)
- [AWS contribuições da comunidade](#)

Cenários

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS contribuições da comunidade

Compilar e testar uma aplicação com tecnologia sem servidor

O exemplo de código a seguir mostra como criar e testar uma aplicação com tecnologia sem servidor usando o API Gateway com o Lambda e o DynamoDB.

SDK para .NET

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK .NET.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Exemplos de Aurora usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Aurora.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Cenários](#)

Cenários

Crie um rastreador de itens de trabalho do Aurora Sem Servidor

O exemplo de código a seguir mostra como criar uma aplicação web que rastreia os itens de trabalho em um banco de dados do Amazon Aurora Sem Servidor e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar o AWS SDK para .NET para criar um aplicativo web que rastreia itens de trabalho em um banco de dados Amazon Aurora e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa um front-end criado com o React.js para interagir com um back-end RESTful do .NET.

- Integre um aplicativo web React com AWS serviços.
- Liste, adicione, atualize e exclua itens em uma tabela do Aurora.
- Envie um relatório por e-mail dos itens de trabalho filtrados usando o Amazon SES.
- Implante e gerencie recursos de exemplo com o AWS CloudFormation script incluído.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Aurora
- Amazon RDS
- Serviços de dados do Amazon RDS
- Amazon SES

Exemplos de Auto Scaling usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com Auto Scaling.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

AttachLoadBalancerTargetGroups

O código de exemplo a seguir mostra como usar `AttachLoadBalancerTargetGroups`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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 }
    });
}

```

- Para obter detalhes da API, consulte [AttachLoadBalancerTargetGroups](#) na Referência AWS SDK para .NET da API.

DeleteAutoScalingGroup

O código de exemplo a seguir mostra como usar DeleteAutoScalingGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Atualize o tamanho mínimo de um grupo do Auto Scaling para zero, encerre todas as instâncias no grupo e exclua o grupo.

```

/// <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>
    /// 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;
}
```

- Para obter detalhes da API, consulte [DeleteAutoScalingGroup](#) na Referência AWS SDK para .NET da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de lançamento e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.

- Execute um servidor web Python em cada EC2 instância para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário interativo em um prompt de comando.

```
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;
}

```

Crie uma classe que envolva as ações do Auto Scaling e da Amazon EC2 .

```

/// <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;
    }
}

```

Crie uma classe que envolva ações do 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);
            }
        }
    }
}

```

Crie uma classe que use o DynamoDB para simular um serviço de recomendação.

```

/// <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");
        }
    }
}

```

Crie uma classe que envolva as ações do 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 });
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .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](#)

Exemplos do Amazon Bedrock Runtime usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET Amazon Bedrock Runtime.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Cenários](#)
- [Amazon Nova](#)
- [Amazon Nova Canvas](#)
- [Claude da Anthropic](#)
- [Command da Cohere](#)
- [Llama da Meta](#)
- [Mistral AI](#)

Cenários

Criar um aplicativo playground para interagir com os modelos de base do Amazon Bedrock

O exemplo de código a seguir mostra como criar playgrounds para interagir com os modelos de base do Amazon Bedrock por meio de diferentes modalidades.

SDK para .NET

O .NET Foundation Model (FM) Playground é um aplicativo de amostra do .NET MAUI Blazor que mostra como usar o Amazon Bedrock a partir do código C#. Este exemplo mostra como os desenvolvedores de .NET e C# podem usar o Amazon Bedrock para criar aplicativos habilitados para IA generativa. É possível testar e interagir com os modelos de base do Amazon Bedrock usando os quatro playgrounds a seguir:

- Um playground de texto.
- Um playground de chat.
- Um playground de chat por voz.
- Um playground de imagens.

O exemplo também lista e exibe os modelos de base aos quais você tem acesso e respectivas características. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Bedrock Runtime

Uso de ferramenta com a API Converse

O exemplo de código a seguir mostra como criar uma interação típica entre um aplicativo, um modelo generativo de IA e ferramentas conectadas ou como APIs mediar interações entre a IA e o mundo externo. Ele usa o exemplo de conectar uma API de meteorologia externa ao modelo de IA para que possa fornecer informações de meteorologia em tempo real com base na entrada do usuário.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

A execução primária do fluxo do cenário. Esse cenário orquestra a conversa entre o usuário, a API Converse do Amazon Bedrock e uma ferramenta de meteorologia.

```
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();
    }
}

```


A ferramenta de meteorologia usada pela demonstração. Esse arquivo define a especificação da ferramenta e implementa a lógica para recuperar dados de meteorologia usando a 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;
    }
}
}

```

A ação da API Converse com uma configuração de ferramenta.

```

/// <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;
    }
}
}
```

- Consulte detalhes da API em [Converse](#) na Referência de API do AWS SDK para .NET .

Amazon Nova

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Amazon Nova usando a API Converse do Bedrock.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Envie uma mensagem de texto para o Amazon Nova usando a API Converse do 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;
}
```

Envie uma conversa de mensagens para o Amazon Nova usando a API Converse do Bedrock com uma configuração de ferramenta.

```
/// <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;
}
}
```

- Consulte detalhes da API em [Converse](#) na Referência de API do AWS SDK para .NET .

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto para o Amazon Nova usando a API Converse do Bedrock e processar o fluxo de respostas em tempo real.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Envie uma mensagem de texto para o Amazon Nova usando a API Converse do Bedrock e processe o fluxo de respostas em tempo real.

```
// 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;
    }
}

```

- Para obter detalhes da API, consulte [ConverseStream](#) Referência AWS SDK para .NET da API.

Cenário: uso de ferramentas com a API Converse

O exemplo de código a seguir mostra como criar uma interação típica entre um aplicativo, um modelo generativo de IA e ferramentas conectadas ou como APIs mediar interações entre a IA e o mundo externo. Ele usa o exemplo de conectar uma API de meteorologia externa ao modelo de IA para que possa fornecer informações de meteorologia em tempo real com base na entrada do usuário.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

A execução primária do fluxo do cenário. Esse cenário orquestra a conversa entre o usuário, a API Converse do Amazon Bedrock e uma ferramenta de meteorologia.

```
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 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
=====
");
    }

    /// <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();
    }
}

```

A ferramenta de meteorologia usada pela demonstração. Esse arquivo define a especificação da ferramenta e implementa a lógica para recuperar dados de meteorologia usando a 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;
    }
}
}

```

A ação da API Converse com uma configuração de ferramenta.

```

/// <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;
    }
}
}
```

- Consulte detalhes da API em [Converse](#) na Referência de API do AWS SDK para .NET .

Amazon Nova Canvas

InvokeModel

O exemplo de código a seguir mostra como invocar o Amazon Nova Canvas no Amazon Bedrock para gerar uma imagem.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie uma imagem com o 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;
}
```

- Para obter detalhes da API, consulte [InvokeModel](#) a Referência AWS SDK para .NET da API.

Claude da Anthropic

InvokeModelWithResponseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto para modelos Claude da Anthropic usando a API Invoke Model e imprimir o fluxo de resposta.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// 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;
}
```

- Para obter detalhes da API, consulte [InvokeModelWithResponseStream](#) a Referência AWS SDK para .NET da API.

Command da Cohere

InvokeModel: Comando R e R+

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Cohere Command R e R+ usando a API InvokeModel.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto.

```
// 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;
}
```

- Para obter detalhes da API, consulte [InvokeModel](#) na Referência AWS SDK para .NET da API.

InvokeModelWithResponseStream: Comando R e R+

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Cohere Command, usando a API InvokeModel com um fluxo de resposta.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// 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;
}
```


- Para obter detalhes da API, consulte [InvokeModel](#) a Referência AWS SDK para .NET da API.

Llama da Meta

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Llama da Meta usando a API Invoke Model.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto.

```
// 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.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 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;
}

```

- Para obter detalhes da API, consulte [InvokeModel](#) a Referência AWS SDK para .NET da API.

InvokeModelWithResponseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Llama da Meta usando a API Invoke Model e imprimir o fluxo de resposta.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// 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;
}

```

- Para obter detalhes da API, consulte [InvokeModelWithResponseStream](#) a Referência AWS SDK para .NET da API.

Mistral AI

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto aos modelos da Mistral usando a API Invoke Model.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto.

```
// 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;
}

```

- Para obter detalhes da API, consulte [InvokeModel](#) a Referência AWS SDK para .NET da API.

InvokeModelWithResponseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto aos modelos da Mistral AI, usando a API `InvokeModel`, e imprimir o fluxo de resposta.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use a API InvokeModel para enviar uma mensagem de texto e processar o fluxo de resposta em tempo real.

```
// 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;
}
```

- Para obter detalhes da API, consulte [InvokeModelWithResponseStream](#) na Referência AWS SDK para .NET da API.

CloudWatch Exemplos de registros usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with CloudWatch Logs.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

AssociateKmsKey

O código de exemplo a seguir mostra como usar AssociateKmsKey.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}.");
}
}
```

- Para obter detalhes da API, consulte [AssociateKmsKey](#) na Referência AWS SDK para .NET da API.

CancelExportTask

O código de exemplo a seguir mostra como usar `CancelExportTask`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
        }
    }
}
```

- Para obter detalhes da API, consulte [CancelExportTask](#) a Referência AWS SDK para .NET da API.

CreateExportTask

O código de exemplo a seguir mostra como usar CreateExportTask.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
    }
}
}

```

- Para obter detalhes da API, consulte [CreateExportTask](#) na Referência AWS SDK para .NET da API.

CreateLogGroup

O código de exemplo a seguir mostra como usar CreateLogGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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.");
    }
}
}
```

- Para obter detalhes da API, consulte [CreateLogGroup](#) Referência AWS SDK para .NET da API.

CreateLogStream

O código de exemplo a seguir mostra como usar CreateLogStream.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
        }
    }
}
```

- Para obter detalhes da API, consulte [CreateLogStream](#) a Referência AWS SDK para .NET da API.

DeleteLogGroup

O código de exemplo a seguir mostra como usar DeleteLogGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}.");
        }
    }
}
```

- Para obter detalhes da API, consulte [DeleteLogGroup](#) Referência AWS SDK para .NET da API.

DescribeExportTasks

O código de exemplo a seguir mostra como usar `DescribeExportTasks`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}
```

- Para obter detalhes da API, consulte [DescribeExportTasks](#) na Referência AWS SDK para .NET da API.

DescribeLogGroups

O código de exemplo a seguir mostra como usar `DescribeLogGroups`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}

```

- Para obter detalhes da API, consulte [DescribeLogGroups](#) na Referência AWS SDK para .NET da API.

StartLiveTail

O código de exemplo a seguir mostra como usar StartLiveTail.

SDK para .NET

Inclua os arquivos necessários.

```

using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

```

Inicie a sessão do 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;
}
```

Você pode lidar com os eventos da sessão do Live Tail de duas maneiras:

```
/* 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");
}
```

- Para obter detalhes da API, consulte [StartLiveTail](#) Referência AWS SDK para .NET da API.

Exemplos de provedores de identidade do Amazon Cognito usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET Amazon Cognito Identity Provider.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

AdminGetUser

O código de exemplo a seguir mostra como usar AdminGetUser.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [AdminGetUser](#) Referência AWS SDK para .NET da API.

AdminInitiateAuth

O código de exemplo a seguir mostra como usar AdminInitiateAuth.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [AdminInitiateAuth](#) Referência AWS SDK para .NET da API.

AdminRespondToAuthChallenge

O código de exemplo a seguir mostra como usar `AdminRespondToAuthChallenge`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }

```

- Para obter detalhes da API, consulte [AdminRespondToAuthChallenge](#) Referência AWS SDK para .NET da API.

AssociateSoftwareToken

O código de exemplo a seguir mostra como usar AssociateSoftwareToken.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;  
    }  
}
```

- Para obter detalhes da API, consulte [AssociateSoftwareToken](#) na Referência AWS SDK para .NET da API.

ConfirmDevice

O código de exemplo a seguir mostra como usar ConfirmDevice.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [ConfirmDevice](#) a Referência AWS SDK para .NET da API.

ConfirmSignUp

O código de exemplo a seguir mostra como usar ConfirmSignUp.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [ConfirmSignUp](#) na Referência AWS SDK para .NET da API.

InitiateAuth

O código de exemplo a seguir mostra como usar InitiateAuth.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [InitiateAuth](#) Referência AWS SDK para .NET da API.

ListUsers

O código de exemplo a seguir mostra como usar ListUsers.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

```
}
```

- Para obter detalhes da API, consulte [ListUsers](#) a Referência AWS SDK para .NET da API.

ResendConfirmationCode

O código de exemplo a seguir mostra como usar ResendConfirmationCode.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```


- Para obter detalhes da API, consulte [ResendConfirmationCode](#) na Referência AWS SDK para .NET da API.

SignUp

O código de exemplo a seguir mostra como usar SignUp.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [SignUp](#) Referência AWS SDK para .NET da API.

VerifySoftwareToken

O código de exemplo a seguir mostra como usar VerifySoftwareToken.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [VerifySoftwareToken](#) na Referência AWS SDK para .NET da API.

Cenários

Inscriver um usuário em um grupo de usuários que exija MFA

O exemplo de código a seguir mostra como:

- Inscriver e confirmar um usuário com nome de usuário, senha e endereço de e-mail.
- Configurar a autenticação multifator associando uma aplicação de MFA ao usuário.
- Faça login usando uma senha e um código de MFA.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"\\nSigning 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.Write("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.Write("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;
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)

- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Exemplos do Amazon Comprehend usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Comprehend.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

DetectDominantLanguage

O código de exemplo a seguir mostra como usar DetectDominantLanguage.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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");  
}  
}
```

- Para obter detalhes da API, consulte [DetectDominantLanguage](#) a Referência AWS SDK para .NET da API.

DetectEntities

O código de exemplo a seguir mostra como usar DetectEntities.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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");
}
```

- Para obter detalhes da API, consulte [DetectEntities](#) na Referência AWS SDK para .NET da API.

DetectKeyPhrases

O código de exemplo a seguir mostra como usar DetectKeyPhrases.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.USWest2);

        // 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");
    }
}

```

- Para obter detalhes da API, consulte [DetectKeyPhrases](#) a Referência AWS SDK para .NET da API.

DetectPiiEntities

O código de exemplo a seguir mostra como usar DetectPiiEntities.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
            }
        }
    }
}

```

- Para obter detalhes da API, consulte [DetectPiiEntities](#) a Referência AWS SDK para .NET da API.

DetectSentiment

O código de exemplo a seguir mostra como usar DetectSentiment.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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");
}
}
```

- Para obter detalhes da API, consulte [DetectSentiment](#) Referência AWS SDK para .NET da API.

DetectSyntax

O código de exemplo a seguir mostra como usar DetectSyntax.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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");
    }
}

```

- Para obter detalhes da API, consulte [DetectSyntax](#) Referência AWS SDK para .NET da API.

StartTopicsDetectionJob

O código de exemplo a seguir mostra como usar StartTopicsDetectionJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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 method 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}");
}
}
```

- Para obter detalhes da API, consulte [StartTopicsDetectionJob](#) Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação para analisar o feedback dos clientes

O exemplo de código a seguir mostra como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

SDK para .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemplos do Amazon DocumentDB usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon DocumentDB.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Exemplos sem servidor](#)

Exemplos sem servidor

Invocar uma função do Lambda de um acionador do Amazon DocumentDB

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo de alterações do DocumentDB. A função recupera a carga útil do DocumentDB e registra em log o conteúdo do registro.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Processando um evento do Amazon DocumentDB com o Lambda ao usar .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; }
    }
```

```
}  
}
```

Exemplos do DynamoDB usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o DynamoDB.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

AWS as contribuições da comunidade são exemplos que foram criados e mantidos por várias equipes AWS. Para deixar seu feedback, use o mecanismo fornecido nos repositórios vinculados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)
- [Exemplos sem servidor](#)
- [AWS contribuições da comunidade](#)

Ações

BatchExecuteStatement

O código de exemplo a seguir mostra como usar BatchExecuteStatement.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use lotes de instruções INSERT para adicionar itens.

```

    /// <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!;
    }
}

```

Use lotes de instruções SELECT para obter itens.

```

/// <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;
}

```

```
    }

}
```

Use lotes de instruções UPDATE para atualizar itens.

```
/// <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;
}

```

Use lotes de instruções DELETE para excluir itens.

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [BatchExecuteStatement](#) na Referência AWS SDK para .NET da API.

BatchGetItem

O código de exemplo a seguir mostra como usar BatchGetItem.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}

```

- Para obter detalhes da API, consulte [BatchGetItem](#) a Referência AWS SDK para .NET da API.

DescribeTable

O código de exemplo a seguir mostra como usar DescribeTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
}
```

- Para obter detalhes da API, consulte [DescribeTable](#) na Referência AWS SDK para .NET da API.

ExecuteStatement

O código de exemplo a seguir mostra como usar ExecuteStatement.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Use uma instrução INSERT para adicionar um item.

```
/// <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;
        }

```

Use uma instrução SELECT para obter um item.

```

/// <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;
}

```

Use uma instrução SELECT para obter uma lista de itens.

```

/// <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;
}

```

Use uma instrução UPDATE para atualizar um item.

```

    /// <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;
    }

```

Use uma instrução DELETE para excluir um único filme.

```

    /// <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;
    }

```

- Para obter detalhes da API, consulte [ExecuteStatement](#) na Referência AWS SDK para .NET da API.

ListTables

O código de exemplo a seguir mostra como usar ListTables.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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);
}
```

- Para obter detalhes da API, consulte [ListTables](#) na Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Criar uma aplicação Web para monitorar dados do DynamoDB

O exemplo de código a seguir mostra como criar uma aplicação Web que monitora itens de trabalho em uma tabela do Amazon DynamoDB e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar a API .NET do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Consultar uma tabela usando lotes de instruções PartiQL

O exemplo de código a seguir mostra como:

- Obter um lote de itens executando várias instruções SELECT.
- Adicionar um lote de itens executando várias instruções INSERT.
- Atualizar um lote de itens executando várias instruções UPDATE.
- Excluir um lote de itens executando várias instruções DELETE.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
// 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;
    }

```

- Para obter detalhes da API, consulte [BatchExecuteStatement](#) na Referência AWS SDK para .NET da API.

Consultar uma tabela usando o PartiQL

O exemplo de código a seguir mostra como:

- Obter um item executando uma instrução SELECT.
- Adicionar um item executando uma instrução INSERT.
- Atualizar um item executando a instrução UPDATE.
- Excluir um item executando uma instrução DELETE.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }

```

- Para obter detalhes da API, consulte [ExecuteStatement](#) na Referência AWS SDK para .NET da API.

Usar um modelo de documento

O exemplo de código a seguir mostra como realizar operações de criação, leitura, atualização e exclusão (CRUD) e em lote usando um modelo de documento para o DynamoDB e um SDK. AWS

Para obter mais informações, consulte o [modelo de documento](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute operações CRUD usando um modelo de documento.

```
/// <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.ExpressionAttributeValueValues[":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);
        }
    }
}

```

Execute operações de gravação em lote usando um modelo de documento.

```

/// <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();
    }
}

```

Verifique uma tabela usando um modelo de documento.

```
/// <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}");
    }
}
}

```

Consulte e verifique uma tabela usando um modelo de documento.

```

/// <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 bane 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}");
    }
}
}

```

Usar um modelo de persistência de objetos de alto nível

O exemplo de código a seguir mostra como realizar operações de criação, leitura, atualização e exclusão (CRUD) e em lote usando um modelo de persistência de objetos para o DynamoDB e um SDK. AWS

Para obter mais informações, consulte [Modelo de persistência de objetos](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Realize operações CRUD usando um modelo de persistência de objetos de alto nível.

```

/// <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.SaveAsync(myBook);

        // 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.SaveAsync(bookRetrieved);

        // 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");
        }
    }
}

```

Realize operações de gravação em lote usando um modelo de persistência de objetos de alto nível.

```

/// <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);
    }
}

```

Mapeie dados arbitrários em uma tabela usando um modelo de persistência de objetos de alto nível.

```

/// <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);
    }
}

```

Consulte e verifique uma tabela usando um modelo de persistência de objetos de alto nível.

```

/// <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}");
}
}
```

Exemplos sem servidor

Invocar uma função do Lambda em um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Como consumir um evento do DynamoDB com o Lambda usando .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.");
    }
}
```

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um fluxo do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando .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 contribuições da comunidade

Compilar e testar uma aplicação com tecnologia sem servidor

O exemplo de código a seguir mostra como criar e testar uma aplicação com tecnologia sem servidor usando o API Gateway com o Lambda e o DynamoDB.

SDK para .NET

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK .NET.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

EC2 Exemplos da Amazon usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com a Amazon EC2.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um par de chaves e um grupo de segurança.
- Selecionar uma imagem de máquina da Amazon (AMI) e um tipo de instância compatível e, em seguida, criar uma instância.
- Interromper e reiniciar a instância.
- Associar um endereço IP elástico à sua instância.
- Conectar-se à sua instância via SSH e, em seguida, limpar os recursos.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário em um prompt de comando.

```
/// <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($"\\t{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);
    }
}

```

Defina uma classe que envolva EC2 ações.

```

/// <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($"InstanceType: {instance.InstanceType}");
        Console.WriteLine($"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($"Showing 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.WriteLine($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\\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.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>
/// 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($"The 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;
}

}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateKeyPair](#)
 - [CreateSecurityGroup](#)
 - [DeleteKeyPair](#)
 - [DeleteSecurityGroup](#)
 - [DescribeImages](#)
 - [DescribeInstanceTypes](#)
 - [DescribeInstances](#)
 - [DescribeKeyPairs](#)
 - [DescribeSecurityGroups](#)
 - [DisassociateAddress](#)
 - [ReleaseAddress](#)
 - [RunInstances](#)
 - [StartInstances](#)
 - [StopInstances](#)
 - [TerminateInstances](#)
 - [UnmonitorInstances](#)

Ações

AllocateAddress

O código de exemplo a seguir mostra como usar `AllocateAddress`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}

```

- Para obter detalhes da API, consulte [AllocateAddress](#) a Referência AWS SDK para .NET da API.

AssociateAddress

O código de exemplo a seguir mostra como usar AssociateAddress.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
}
```

- Para obter detalhes da API, consulte [AssociateAddress](#) Referência AWS SDK para .NET da API.

AuthorizeSecurityGroupIngress

O código de exemplo a seguir mostra como usar AuthorizeSecurityGroupIngress.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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();
}

```

- Para obter detalhes da API, consulte [AuthorizeSecurityGroupIngress](#) a Referência AWS SDK para .NET da API.

CreateKeyPair

O código de exemplo a seguir mostra como usar `CreateKeyPair`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }

```

- Para obter detalhes da API, consulte [CreateKeyPair](#) Referência AWS SDK para .NET da API.

CreateLaunchTemplate

O código de exemplo a seguir mostra como usar CreateLaunchTemplate.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}
}

```

- Para obter detalhes da API, consulte [CreateLaunchTemplate](#) a Referência AWS SDK para .NET da API.

CreateSecurityGroup

O código de exemplo a seguir mostra como usar CreateSecurityGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
        }
    }
}
```

- Para obter detalhes da API, consulte [CreateSecurityGroup](#) Referência AWS SDK para .NET da API.

DeleteKeyPair

O código de exemplo a seguir mostra como usar DeleteKeyPair.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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);
    }
}

```

- Para obter detalhes da API, consulte [DeleteKeyPair](#) a Referência AWS SDK para .NET da API.

DeleteLaunchTemplate

O código de exemplo a seguir mostra como usar DeleteLaunchTemplate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}
```

- Para obter detalhes da API, consulte [DeleteLaunchTemplate](#) a Referência AWS SDK para .NET da API.

DeleteSecurityGroup

O código de exemplo a seguir mostra como usar DeleteSecurityGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}

```

- Para obter detalhes da API, consulte [DeleteSecurityGroup](#) Referência AWS SDK para .NET da API.

DescribeAvailabilityZones

O código de exemplo a seguir mostra como usar `DescribeAvailabilityZones`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }
}

```

- Para obter detalhes da API, consulte [DescribeAvailabilityZones](#) na Referência AWS SDK para .NET da API.

DescribeIamInstanceProfileAssociations

O código de exemplo a seguir mostra como usar `DescribeIamInstanceProfileAssociations`.
SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }
}
```

- Para obter detalhes da API, consulte [DescribeIamInstanceProfileAssociations](#) a Referência AWS SDK para .NET da API.

DescribeInstanceTypes

O código de exemplo a seguir mostra como usar `DescribeInstanceTypes`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }
}
```

- Para obter detalhes da API, consulte [DescribeInstanceTypes](#) a Referência AWS SDK para .NET da API.

DescribeInstances

O código de exemplo a seguir mostra como usar `DescribeInstances`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}

```

- Para obter detalhes da API, consulte [DescribeInstances](#) a Referência AWS SDK para .NET da API.

DescribeKeyPairs

O código de exemplo a seguir mostra como usar DescribeKeyPairs.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
        }
    }
}
```

- Para obter detalhes da API, consulte [DescribeKeyPairs](#) a Referência AWS SDK para .NET da API.

DescribeSecurityGroups

O código de exemplo a seguir mostra como usar `DescribeSecurityGroups`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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($"\\tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"{range.CidrIp}
"); });

        Console.WriteLine($"\\n\\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.WriteLine($"{range.CidrIpv6} "); });

        Console.WriteLine($"\\n\\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{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($"\\tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"{range.CidrIp}
"); });

        Console.WriteLine($"\\n\\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.WriteLine($"{range.CidrIpv6} "); });

        Console.WriteLine($"\\n\\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{id.Id} "));

        Console.WriteLine($"\\n\\tTo Port: {permission.ToPort}");
    });
}

```

- Para obter detalhes da API, consulte [DescribeSecurityGroups](#) a Referência AWS SDK para .NET da API.

DescribeSubnets

O código de exemplo a seguir mostra como usar DescribeSubnets.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
        }
    }
}

```

- Para obter detalhes da API, consulte [DescribeSubnets](#) a Referência AWS SDK para .NET da API.

DescribeVpcs

O código de exemplo a seguir mostra como usar DescribeVpcs.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
}
```

- Para obter detalhes da API, consulte [DescribeVpcs](#) a Referência AWS SDK para .NET da API.

DisassociateAddress

O código de exemplo a seguir mostra como usar `DisassociateAddress`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }
}

```

- Para obter detalhes da API, consulte [DisassociateAddress](#) Referência AWS SDK para .NET da API.

RebootInstances

O código de exemplo a seguir mostra como usar RebootInstances.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Reinicialize uma instância pelo ID.

```
/// <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;
    }
}

```

Substitua o perfil de uma instância, reinicialize e reinicie um servidor 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;
}
}

```

- Para obter detalhes da API, consulte [RebootInstances](#) a Referência AWS SDK para .NET da API.

ReleaseAddress

O código de exemplo a seguir mostra como usar ReleaseAddress.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}

```

- Para obter detalhes da API, consulte [ReleaseAddressa](#) Referência AWS SDK para .NET da API.

ReplaceIamInstanceProfileAssociation

O código de exemplo a seguir mostra como usar `ReplaceIamInstanceProfileAssociation`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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" }
                }
            }
        });
        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;
    }
}

```

- Para obter detalhes da API, consulte [ReplacelamInstanceProfileAssociation](#) na Referência AWS SDK para .NET da API.

RunInstances

O código de exemplo a seguir mostra como usar RunInstances.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
        }
    }
}
```

- Para obter detalhes da API, consulte [RunInstances](#) a Referência AWS SDK para .NET da API.

StartInstances

O código de exemplo a seguir mostra como usar StartInstances.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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.WriteLine(". ");
} while (!hasState);

return hasState;
}
```

- Para obter detalhes da API, consulte [StartInstances](#) a Referência AWS SDK para .NET da API.

StopInstances

O código de exemplo a seguir mostra como usar StopInstances.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [StopInstances](#) a Referência AWS SDK para .NET da API.

TerminateInstances

O código de exemplo a seguir mostra como usar `TerminateInstances`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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($"\\n\\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;
}
```

- Para obter detalhes da API, consulte [TerminateInstances](#) a Referência AWS SDK para .NET da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de lançamento e para manter o número de instâncias em um intervalo especificado.

- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor web Python em cada EC2 instância para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário interativo em um prompt de comando.

```
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.LoadB
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.TryTerminateInstanceId(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;
}

```

Crie uma classe que envolva as ações do Auto Scaling e da Amazon EC2 .

```

/// <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 newKeyName)
{
    try
    {
        var keyResponse = await _amazonEc2.CreateKeyPairAsync(
            new CreateKeyPairRequest() { KeyName = newKeyName });
        await File.WriteAllTextAsync($"{newKeyName}.pem",
            keyResponse.KeyPair.KeyMaterial);
        Console.WriteLine($"Created key pair {newKeyName}.");
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine("Key pair already exists.");
    }
}

/// <summary>
/// Delete the key pair and file by name.
/// </summary>
/// <param name="deleteKeyName">The key pair to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteKeyPairByName(string deleteKeyName)
{
    try
    {
        await _amazonEc2.DeleteKeyPairAsync(
            new DeleteKeyPairRequest() { KeyName = deleteKeyName });
        File.Delete($"{deleteKeyName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyName} 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;
    }
}

```

Crie uma classe que envolva ações do 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);
            }
        }
    }
}

```

Crie uma classe que use o DynamoDB para simular um serviço de recomendação.

```

/// <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");
        }
    }
}

```

Crie uma classe que envolva as ações do 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 });
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .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](#)

Exemplos do Amazon ECS usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon ECS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto **em seus cenários relacionados**.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

ListClusters

O código de exemplo a seguir mostra como usar `ListClusters`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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}");
}
}
```

- Para obter detalhes da API, consulte [ListClusters](#) a Referência AWS SDK para .NET da API.

ListServices

O código de exemplo a seguir mostra como usar ListServices.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListServices](#) a Referência AWS SDK para .NET da API.

ListTasks

O código de exemplo a seguir mostra como usar ListTasks.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListTasks](#) a Referência AWS SDK para .NET da API.

Cenários

Obter informações de ARN para clusters, serviços e tarefas

O exemplo de código a seguir mostra como:

- Obter uma lista de todos os clusters.
- Obter serviços para um cluster.
- Obter tarefas para um cluster.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
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));
}
}

```

Métodos de encapsulamento que são chamados pelo cenário para gerenciar as ações do 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;
}
}

```


- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [ListClusters](#)
 - [ListServices](#)
 - [ListTasks](#)

Elastic Load Balancing - Exemplos da versão 2 usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Elastic Load Balancing - Versão 2.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

CreateListener

O código de exemplo a seguir mostra como usar CreateListener.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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];
}

```

- Para obter detalhes da API, consulte [CreateListener](#) na Referência AWS SDK para .NET da API.

CreateLoadBalancer

O código de exemplo a seguir mostra como usar CreateLoadBalancer.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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];
}

```

- Para obter detalhes da API, consulte [CreateLoadBalancer](#) na Referência AWS SDK para .NET da API.

CreateTargetGroup

O código de exemplo a seguir mostra como usar CreateTargetGroup.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [CreateTargetGroup](#) na Referência AWS SDK para .NET da API.

DeleteLoadBalancer

O código de exemplo a seguir mostra como usar DeleteLoadBalancer.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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.");
    }
}

```

- Para obter detalhes da API, consulte [DeleteLoadBalancer](#) na Referência AWS SDK para .NET da API.

DeleteTargetGroup

O código de exemplo a seguir mostra como usar DeleteTargetGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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 }
                    }
                );
        }
        catch { }
    }
}

```



```

        });

        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);
    }
}
}

```

- Para obter detalhes da API, consulte [DeleteTargetGroup](#) na Referência AWS SDK para .NET da API.

DescribeLoadBalancers

O código de exemplo a seguir mostra como usar DescribeLoadBalancers.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [DescribeLoadBalancers](#) na Referência AWS SDK para .NET da API.

DescribeTargetHealth

O código de exemplo a seguir mostra como usar DescribeTargetHealth.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}
```

- Para obter detalhes da API, consulte [DescribeTargetHealth](#) a Referência AWS SDK para .NET da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de lançamento e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.
- Execute um servidor web Python em cada EC2 instância para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário interativo em um prompt de comando.

```
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.TargetGroupName,
            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($"\\nNow, 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;
    }

```

Crie uma classe que envolva as ações do Auto Scaling e da Amazon EC2 .

```

/// <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;
}

///

```

```

/// 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;
    }
}

```

Crie uma classe que envolva ações do 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);
            }
        }
    }
}

```

```
}
```

Crie uma classe que use o DynamoDB para simular um serviço de recomendação.

```
/// <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.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");
        }
    }
}

```

Crie uma classe que envolva as ações do 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 });
}
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .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 exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with EventBridge.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá EventBridge

O exemplo de código a seguir mostra como começar a usar o EventBridge.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
        }
    }
}
```

- Para obter detalhes da API, consulte [ListEventBuses](#) a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar uma regra e adicionar um destino a ela.
- Habilitar e desabilitar regras.
- Listar e atualizar regras e destinos.

- Envie eventos e, em seguida, limpe os recursos.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
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 IAMIdentityManagementService? _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($"Added 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($"{t.Target: {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($"{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;
}
}

```

Crie uma classe que envolva as EventBridge operações.

```

/// <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;
}
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [DeleteRule](#)
 - [DescribeRule](#)
 - [DisableRule](#)
 - [EnableRule](#)
 - [ListRuleNamesByTarget](#)
 - [ListRules](#)
 - [ListTargetsByRule](#)
 - [PutEvents](#)
 - [PutRule](#)
 - [PutTargets](#)

Ações

DeleteRule

O código de exemplo a seguir mostra como usar DeleteRule.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Excluir uma regra pelo nome.

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteRule](#) a Referência AWS SDK para .NET da API.

DescribeRule

O código de exemplo a seguir mostra como usar DescribeRule.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Obter o estado de uma regra usando a descrição da regra.

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeRule](#) a Referência AWS SDK para .NET da API.

DisableRule

O código de exemplo a seguir mostra como usar `DisableRule`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Desabilitar uma regra pelo nome da regra.

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DisableRule](#) a Referência AWS SDK para .NET da API.

EnableRule

O código de exemplo a seguir mostra como usar EnableRule.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Habilitar uma regra pelo nome da regra.


```

/// <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;
}

```

- Para obter detalhes da API, consulte [EnableRule](#) a Referência AWS SDK para .NET da API.

ListRuleNamesByTarget

O código de exemplo a seguir mostra como usar ListRuleNamesByTarget.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Liste todos os nomes das regras usando o destino.

```

/// <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;
}

```

- Para obter detalhes da API, consulte [ListRuleNamesByTarget](#) na Referência AWS SDK para .NET da API.

ListRules

O código de exemplo a seguir mostra como usar `ListRules`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Liste todas as regras para um barramento de eventos.

```

/// <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;
}

```

- Para obter detalhes da API, consulte [ListRules](#) a Referência AWS SDK para .NET da API.

ListTargetsByRule

O código de exemplo a seguir mostra como usar `ListTargetsByRule`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Listar todos os destinos para uma regra usando o nome da regra.

```

/// <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;
}
```

- Para obter detalhes da API, consulte [ListTargetsByRule](#) a Referência AWS SDK para .NET da API.

PutEvents

O código de exemplo a seguir mostra como usar PutEvents.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Envie um evento que corresponda a um padrão personalizado para uma regra.

```
/// <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;
}
```

- Para obter detalhes da API, consulte [PutEvents](#) na Referência AWS SDK para .NET da API.

PutRule

O código de exemplo a seguir mostra como usar PutRule.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie uma regra que seja acionada quando um objeto é adicionado a um bucket do 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;
    }

```

Crie uma regra que utilize um padrão personalizado.

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [PutRule](#) a Referência AWS SDK para .NET da API.

PutTargets

O código de exemplo a seguir mostra como usar PutTargets.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Adicione um tópico do Amazon SNS como um destino para uma regra.

```

/// <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;
}

```

Adicione um transformador de entrada a um destino para uma regra.

```

/// <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;
    }
```

- Para obter detalhes da API, consulte [PutTargets](#) na Referência AWS SDK para .NET da API.

RemoveTargets

O código de exemplo a seguir mostra como usar `RemoveTargets`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Remover todos os destinos de uma regra usando o nome da regra.

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [RemoveTargets](#) na Referência AWS SDK para .NET da API.

EventBridge Exemplos de agendador usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with EventBridge Scheduler.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)

- [Ações](#)
- [Cenários](#)

Conceitos básicos

Olá, EventBridge Agendador

O exemplo de código a seguir mostra como começar a usar o EventBridge Scheduler.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"\\tTotal of {results.Count} schedule(s) available.");
    results.ForEach(s => Console.WriteLine($"\\tSchedule: {s.Name}"));
}
}

```

- Para obter detalhes da API, consulte [ListSchedules](#) a Referência AWS SDK para .NET da API.

Ações

CreateSchedule

O código de exemplo a seguir mostra como usar CreateSchedule.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }
}

```

- Para obter detalhes da API, consulte [CreateSchedule](#) a Referência AWS SDK para .NET da API.

CreateScheduleGroup

O código de exemplo a seguir mostra como usar CreateScheduleGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
        }
    }
}

```

- Para obter detalhes da API, consulte [CreateScheduleGroup](#) Referência AWS SDK para .NET da API.

DeleteSchedule

O código de exemplo a seguir mostra como usar DeleteSchedule.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}

```

- Para obter detalhes da API, consulte [DeleteSchedule](#) na Referência AWS SDK para .NET da API.

DeleteScheduleGroup

O código de exemplo a seguir mostra como usar DeleteScheduleGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
}
```

- Para obter detalhes da API, consulte [DeleteScheduleGroup](#) Referência AWS SDK para .NET da API.

Cenários

Eventos agendados

O exemplo de código a seguir mostra como:

- Implante uma CloudFormation pilha com os recursos necessários.
- Crie um grupo de EventBridge agendamento do Scheduler.
- Crie uma EventBridge agenda única do Scheduler com uma janela de horário flexível.
- Crie uma programação recorrente do EventBridge Scheduler com uma taxa especificada.
- Exclua o EventBridge agendador, a agenda e o grupo de agendamentos.
- Limpe os recursos e exclua a pilha.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário.

```
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}' " +
            $"\\nto 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;
        }
    }
}

```

Wrapper para operações de serviço.

```

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;
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateSchedule](#)
 - [CreateScheduleGroup](#)
 - [DeleteSchedule](#)
 - [DeleteScheduleGroups](#)

Exemplos do Amazon Glacier usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Glacier.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá, Amazon Glacier

O exemplo de código a seguir mostra como começar a usar o Amazon Glacier.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}
```

- Para obter detalhes da API, consulte [ListVaults](#) a Referência AWS SDK para .NET da API.

Ações

AddTagsToVault

O código de exemplo a seguir mostra como usar AddTagsToVault.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [AddTagsToVault](#) Referência AWS SDK para .NET da API.

CreateVault

O código de exemplo a seguir mostra como usar CreateVault.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateVault](#) Referência AWS SDK para .NET da API.

DescribeVault

O código de exemplo a seguir mostra como usar `DescribeVault`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeVaulta](#) Referência AWS SDK para .NET da API.

InitiateJob

O código de exemplo a seguir mostra como usar `InitiateJob`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Recupere um arquivo de um cofre. Este exemplo usa a `ArchiveTransferManager` classe. Para obter detalhes da API, consulte [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}%");
    }
}

```

- Para obter detalhes da API, consulte [InitiateJob](#) Referência AWS SDK para .NET da API.

ListJobs

O código de exemplo a seguir mostra como usar ListJobs.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [ListJobs](#) a Referência AWS SDK para .NET da API.

ListTagsForVault

O código de exemplo a seguir mostra como usar `ListTagsForVault`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [ListTagsForVault](#) Referência AWS SDK para .NET da API.

ListVaults

O código de exemplo a seguir mostra como usar ListVaults.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [ListVaults](#) na Referência AWS SDK para .NET da API.

UploadArchive

O código de exemplo a seguir mostra como usar UploadArchive.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }
}
```

- Para obter detalhes da API, consulte [UploadArchive](#) a Referência AWS SDK para .NET da API.

AWS Glue exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with AWS Glue.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá AWS Glue

O exemplo de código a seguir mostra como começar a usar o AWS Glue.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}
```

- Para obter detalhes da API, consulte [ListJobs](#) a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um crawler que rastreie um bucket público do Amazon S3 e gere um banco de dados de metadados formatado em CSV.
- Liste informações sobre bancos de dados e tabelas em seu AWS Glue Data Catalog.
- Criar um trabalho para extrair dados em CSV do bucket do S3, transformá-los e carregar a saída formatada em JSON em outro bucket do S3.
- Listar informações sobre execuções de tarefas, visualizar dados transformados e limpar recursos.

Para obter mais informações, consulte [Tutorial: Introdução ao AWS Glue Studio](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie uma classe que envolva as AWS Glue funções usadas no cenário.

```
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;
}
}

```

Crie uma classe que execute o cenário.

```

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);
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Ações

CreateCrawler

O código de exemplo a seguir mostra como usar CreateCrawler.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [CreateCrawler](#) na Referência AWS SDK para .NET da API.

CreateJob

O código de exemplo a seguir mostra como usar CreateJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [CreateJob](#) Referência AWS SDK para .NET da API.

DeleteCrawler

O código de exemplo a seguir mostra como usar DeleteCrawler.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteCrawler](#) a Referência AWS SDK para .NET da API.

DeleteDatabase

O código de exemplo a seguir mostra como usar DeleteDatabase.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteDatabase](#) a Referência AWS SDK para .NET da API.

DeleteJob

O código de exemplo a seguir mostra como usar DeleteJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteJob](#) Referência AWS SDK para .NET da API.

DeleteTable

O código de exemplo a seguir mostra como usar DeleteTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [DeleteTable](#) Referência AWS SDK para .NET da API.

GetCrawler

O código de exemplo a seguir mostra como usar `GetCrawler`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [GetCrawler](#) na Referência AWS SDK para .NET da API.

GetDatabase

O código de exemplo a seguir mostra como usar GetDatabase.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;  
}
```

- Para obter detalhes da API, consulte [GetDatabase](#) a Referência AWS SDK para .NET da API.

GetJobRun

O código de exemplo a seguir mostra como usar GetJobRun.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [GetJobRun](#) a Referência AWS SDK para .NET da API.

GetJobRuns

O código de exemplo a seguir mostra como usar GetJobRuns.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetJobRuns](#) na Referência AWS SDK para .NET da API.

GetTables

O código de exemplo a seguir mostra como usar `GetTables`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetTables](#) a Referência AWS SDK para .NET da API.

ListJobs

O código de exemplo a seguir mostra como usar `ListJobs`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListJobs](#) a Referência AWS SDK para .NET da API.

StartCrawler

O código de exemplo a seguir mostra como usar StartCrawler.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).


```
/// <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;
}
```

- Para obter detalhes da API, consulte [StartCrawler](#) a Referência AWS SDK para .NET da API.

StartJobRun

O código de exemplo a seguir mostra como usar StartJobRun.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [StartJobRun](#) na Referência AWS SDK para .NET da API.

Exemplos de IAM usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o IAM.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)

Conceitos básicos

Olá, IAM

O exemplo de código a seguir mostra como começar a usar o IAM.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");  
});  
}  
}
```

- Para obter detalhes da API, consulte [ListPolicies](#) na Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como criar um usuário e assumir um perfil.

Warning

Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como [Centro de Identidade do AWS IAM](#).

- Crie um usuário sem permissões.
- Crie uma função que conceda permissão para listar os buckets do Amazon S3 para a conta.
- Adicione uma política para permitir que o usuário assuma a função.
- Assuma o perfil e liste buckets do S3 usando credenciais temporárias, depois limpe os recursos.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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 IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService 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<IAMIdentityManagementService>()
                .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();
}
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Ações

AttachRolePolicy

O código de exemplo a seguir mostra como usar `AttachRolePolicy`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [AttachRolePolicy](#) a Referência AWS SDK para .NET da API.

CreateAccessKey

O código de exemplo a seguir mostra como usar CreateAccessKey.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateAccessKey](#) Referência AWS SDK para .NET da API.

CreateInstanceProfile

O código de exemplo a seguir mostra como usar CreateInstanceProfile.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }
}

```

- Para obter detalhes da API, consulte [CreateInstanceProfile](#) a Referência AWS SDK para .NET da API.

CreatePolicy

O código de exemplo a seguir mostra como usar CreatePolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [CreatePolicy](#) na Referência AWS SDK para .NET da API.

CreateRole

O código de exemplo a seguir mostra como usar CreateRole.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [CreateRole](#) a Referência AWS SDK para .NET da API.

CreateServiceLinkedRole

O código de exemplo a seguir mostra como usar `CreateServiceLinkedRole`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateServiceLinkedRole](#) a Referência AWS SDK para .NET da API.

CreateUser

O código de exemplo a seguir mostra como usar CreateUser.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateUser](#) a Referência AWS SDK para .NET da API.

DeleteAccessKey

O código de exemplo a seguir mostra como usar DeleteAccessKey.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }

```

- Para obter detalhes da API, consulte [DeleteAccessKey](#) a Referência AWS SDK para .NET da API.

DeleteInstanceProfile

O código de exemplo a seguir mostra como usar DeleteInstanceProfile.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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.");
    }
}

```

```
}
```

- Para obter detalhes da API, consulte [DeleteInstanceProfile](#) a Referência AWS SDK para .NET da API.

DeletePolicy

O código de exemplo a seguir mostra como usar DeletePolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeletePolicy](#) a Referência AWS SDK para .NET da API.

DeleteRole

O código de exemplo a seguir mostra como usar DeleteRole.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteRole](#) a Referência AWS SDK para .NET da API.

DeleteRolePolicy

O código de exemplo a seguir mostra como usar DeleteRolePolicy.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }

```

- Para obter detalhes da API, consulte [DeleteRolePolicy](#) a Referência AWS SDK para .NET da API.

DeleteUser

O código de exemplo a seguir mostra como usar DeleteUser.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }
```

- Para obter detalhes da API, consulte [DeleteUser](#) Referência AWS SDK para .NET da API.

DeleteUserPolicy

O código de exemplo a seguir mostra como usar DeleteUserPolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteUserPolicy](#) Referência AWS SDK para .NET da API.

DetachRolePolicy

O código de exemplo a seguir mostra como usar DetachRolePolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DetachRolePolicy](#) a Referência AWS SDK para .NET da API.

GetAccountPasswordPolicy

O código de exemplo a seguir mostra como usar GetAccountPasswordPolicy.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetAccountPasswordPolicy](#) a Referência AWS SDK para .NET da API.

GetPolicy

O código de exemplo a seguir mostra como usar GetPolicy.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetPolicy](#) a Referência AWS SDK para .NET da API.

GetRole

O código de exemplo a seguir mostra como usar GetRole.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetRole](#) a Referência AWS SDK para .NET da API.

GetUser

O código de exemplo a seguir mostra como usar GetUser.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetUser](#) a Referência AWS SDK para .NET da API.

ListAttachedRolePolicies

O código de exemplo a seguir mostra como usar ListAttachedRolePolicies.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListAttachedRolePolicies](#) a Referência AWS SDK para .NET da API.

ListGroups

O código de exemplo a seguir mostra como usar `ListGroups`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListGroups](#) a Referência AWS SDK para .NET da API.

ListPolicies

O código de exemplo a seguir mostra como usar ListPolicies.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [ListPolicies](#) a Referência AWS SDK para .NET da API.

ListRolePolicies

O código de exemplo a seguir mostra como usar ListRolePolicies.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [ListRolePolicies](#) na Referência AWS SDK para .NET da API.

ListRoles

O código de exemplo a seguir mostra como usar ListRoles.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListRoles](#) na Referência AWS SDK para .NET da API.

ListSAMLProviders

O código de exemplo a seguir mostra como usar `ListSAMLProviders`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Lista SAMLProviders](#) na referência AWS SDK para .NET da API.

ListUsers

O código de exemplo a seguir mostra como usar `ListUsers`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListUsers](#) na Referência AWS SDK para .NET da API.

PutRolePolicy

O código de exemplo a seguir mostra como usar PutRolePolicy.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [PutRolePolicy](#) a Referência AWS SDK para .NET da API.

Cenários

Criar e gerenciar um serviço resiliente

O exemplo de código a seguir mostra como criar um serviço web com balanceamento de carga que retorna recomendações de livros, filmes e músicas. O exemplo mostra como o serviço responde a falhas e como é possível reestruturá-lo para gerar mais resiliência em caso de falhas.

- Use um grupo do Amazon EC2 Auto Scaling para criar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) com base em um modelo de lançamento e para manter o número de instâncias em um intervalo especificado.
- Gerencie e distribua solicitações HTTP com o Elastic Load Balancing.
- Monitore a integridade das instâncias em um grupo do Auto Scaling e encaminhe solicitações somente para instâncias íntegras.

- Execute um servidor web Python em cada EC2 instância para lidar com solicitações HTTP. O servidor Web responde com recomendações e verificações de integridade.
- Simule um serviço de recomendação com uma tabela do Amazon DynamoDB.
- Controle a resposta do servidor web às solicitações e verificações de saúde atualizando AWS Systems Manager os parâmetros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário interativo em um prompt de comando.

```
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.LoadB
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;
}

```

Crie uma classe que envolva as ações do Auto Scaling e da Amazon EC2 .

```

/// <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;
    }
}

```

Crie uma classe que envolva ações do 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);
            }
        }
    }
}

```

Crie uma classe que use o DynamoDB para simular um serviço de recomendação.

```

/// <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");
        }
    }
}

```

Crie uma classe que envolva as ações do 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 });
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .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](#)

Exemplos do Amazon Keyspaces usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Keyspaces.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá, Amazon Keyspaces

O exemplo de código a seguir mostra como começar a usar o Amazon Keyspaces.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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)
            )
            .Build();
    }
}
```

```
        .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonKeyspaces>()
        .AddTransient<KeyspacesWrapper>()
    )
    .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<HelloKeyspaces>();

var keyspaceClient = host.Services.GetRequiredService<IAmazonKeyspaces>();
var keyspaceWrapper = new KeyspacesWrapper(keyspaceClient);

Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
await keyspaceWrapper.ListKeyspaces();
    }
}
```

- Para obter detalhes da API, consulte [ListKeyspaces](#) a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um keyspace e uma tabela. O esquema da tabela contém dados do filme e tem a point-in-time recuperação ativada.
- Conectar-se ao keyspace usando uma conexão TLS segura com autenticação SigV4.
- Consultar a tabela. Adicionar, recuperar e atualizar dados do filme.
- Atualizar a tabela. Adicionar uma coluna para rastrear os filmes assistidos.
- Restaure a tabela ao estado anterior e limpe os recursos.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"\\n\\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.Write("Waiting for the new table to be active. ");
    do
    {
        try
        {
            resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
            Console.Write(".");
        }
    }
}
```



```

        }
        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.WriteLine(".");
}
}

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!;
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateKeyspace](#)
 - [CreateTable](#)
 - [DeleteKeyspace](#)
 - [DeleteTable](#)
 - [GetKeyspace](#)
 - [GetTable](#)

- [ListKeyspaces](#)
- [ListTables](#)
- [RestoreTable](#)
- [UpdateTable](#)

Ações

CreateKeyspace

O código de exemplo a seguir mostra como usar CreateKeyspace.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateKeyspace](#) a Referência AWS SDK para .NET da API.

CreateTable

O código de exemplo a seguir mostra como usar CreateTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateTable](#) a Referência AWS SDK para .NET da API.

DeleteKeyspace

O código de exemplo a seguir mostra como usar DeleteKeyspace.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteKeyspace](#) a Referência AWS SDK para .NET da API.

DeleteTable

O código de exemplo a seguir mostra como usar DeleteTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [DeleteTable](#) na Referência AWS SDK para .NET da API.

GetKeyspace

O código de exemplo a seguir mostra como usar GetKeyspace.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [GetKeyspace](#) a Referência AWS SDK para .NET da API.

GetTable

O código de exemplo a seguir mostra como usar `GetTable`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetTable](#) a Referência AWS SDK para .NET da API.

ListKeyspaces

O código de exemplo a seguir mostra como usar `ListKeyspaces`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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}");
    }
}
```

- Para obter detalhes da API, consulte [ListKeyspaces](#) na Referência AWS SDK para .NET da API.

ListTables

O código de exemplo a seguir mostra como usar `ListTables`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [ListTables](#) na Referência AWS SDK para .NET da API.

RestoreTable

O código de exemplo a seguir mostra como usar RestoreTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [RestoreTable](#) a Referência AWS SDK para .NET da API.

UpdateTable

O código de exemplo a seguir mostra como usar UpdateTable.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}
```

- Para obter detalhes da API, consulte [UpdateTable](#) a Referência AWS SDK para .NET da API.

Exemplos de Kinesis usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Kinesis.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Exemplos sem servidor](#)

Ações

AddTagsToStream

O código de exemplo a seguir mostra como usar AddTagsToStream.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}

```

- Para obter detalhes da API, consulte [AddTagsToStream](#) na Referência AWS SDK para .NET da API.

CreateStream

O código de exemplo a seguir mostra como usar CreateStream.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}
```

- Para obter detalhes da API, consulte [CreateStream](#) a Referência AWS SDK para .NET da API.

DeleteStream

O código de exemplo a seguir mostra como usar DeleteStream.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}

```

- Para obter detalhes da API, consulte [DeleteStream](#) na Referência AWS SDK para .NET da API.

DeregisterStreamConsumer

O código de exemplo a seguir mostra como usar `DeregisterStreamConsumer`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}

```

- Para obter detalhes da API, consulte [DeregisterStreamConsumer](#) Referência AWS SDK para .NET da API.

ListStreamConsumers

O código de exemplo a seguir mostra como usar `ListStreamConsumers`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}

```

- Para obter detalhes da API, consulte [ListStreamConsumers](#) na Referência AWS SDK para .NET da API.

ListStreams

O código de exemplo a seguir mostra como usar ListStreams.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).


```
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.");
        }
    }
}
```

- Para obter detalhes da API, consulte [ListStreams](#) na Referência AWS SDK para .NET da API.

ListTagsForStream

O código de exemplo a seguir mostra como usar `ListTagsForStream`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}"));
    }
}

```

- Para obter detalhes da API, consulte [ListTagsForStream](#) Referência AWS SDK para .NET da API.

RegisterStreamConsumer

O código de exemplo a seguir mostra como usar RegisterStreamConsumer.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;  
    }  
}
```

- Para obter detalhes da API, consulte [RegisterStreamConsumer](#) a Referência AWS SDK para .NET da API.

Exemplos sem servidor

Invocar uma função do Lambda em um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um stream do Kinesis. A função recupera a carga útil do Kinesis, decodifica do Base64 e registra o conteúdo do registro em log.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do Kinesis com o Lambda usando .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;
    }
}

```

Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um stream do Kinesis. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Relatar falhas de itens em lote do Kinesis com o Lambda usando o .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 exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with AWS KMS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

CreateAlias

O código de exemplo a seguir mostra como usar CreateAlias.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");  
    }  
}  
}
```

- Para obter detalhes da API, consulte [CreateAlias](#) a Referência AWS SDK para .NET da API.

CreateGrant

O código de exemplo a seguir mostra como usar CreateGrant.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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 = "7c9eccc2-38cb-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}");
}
}
```

- Para obter detalhes da API, consulte [CreateGrant](#) na Referência AWS SDK para .NET da API.

CreateKey

O código de exemplo a seguir mostra como usar `CreateKey`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
            }
        }
    }
}

```

- Para obter detalhes da API, consulte [CreateKey](#) a Referência AWS SDK para .NET da API.

DescribeKey

O código de exemplo a seguir mostra como usar DescribeKey.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [DescribeKey](#) Referência AWS SDK para .NET da API.

DisableKey

O código de exemplo a seguir mostra como usar `DisableKey`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [DisableKey](#) Referência AWS SDK para .NET da API.

EnableKey

O código de exemplo a seguir mostra como usar EnableKey.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
}
}
```

- Para obter detalhes da API, consulte [EnableKey](#) a Referência AWS SDK para .NET da API.

ListAliases

O código de exemplo a seguir mostra como usar ListAliases.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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);
    }
}

```

- Para obter detalhes da API, consulte [ListAliases](#) a Referência AWS SDK para .NET da API.

ListGrants

O código de exemplo a seguir mostra como usar ListGrants.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}
```

- Para obter detalhes da API, consulte [ListGrants](#) a Referência AWS SDK para .NET da API.

ListKeys

O código de exemplo a seguir mostra como usar ListKeys.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
```

- Para obter detalhes da API, consulte [ListKeys](#) a Referência AWS SDK para .NET da API.

Exemplos de Lambda usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Lambda.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

AWS as contribuições da comunidade são exemplos que foram criados e mantidos por várias equipes AWS. Para deixar seu feedback, use o mecanismo fornecido nos repositórios vinculados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)
- [Exemplos sem servidor](#)
- [AWS contribuições da comunidade](#)

Conceitos básicos

Olá, Lambda

O exemplo de código a seguir mostra como começar a usar o Lambda.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");  
    });  
}  
}
```

- Para obter detalhes da API, consulte [ListFunctions](#) Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um perfil do IAM e uma função do Lambda e carregar o código de manipulador.
- Invocar essa função com um único parâmetro e receber resultados.
- Atualizar o código de função e configurar usando uma variável de ambiente.
- Invocar a função com novos parâmetros e receber resultados. Exibir o log de execução retornado.
- Listar as funções para sua conta e limpar os recursos.

Para obter mais informações, consulte [Criar uma função do Lambda no console](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie métodos que executem ações do 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;
    }
}

```

Crie uma função que execute o cenário.

```

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.WriteLine("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.WriteLine("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();
    }
}

```

Defina um manipulador do Lambda que aumente um número.

```

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;
    }
}
}

```

Defina um segundo manipulador do Lambda que faça operações aritméticas.

```

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 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;
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Ações

CreateFunction

O código de exemplo a seguir mostra como usar `CreateFunction`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [CreateFunction](#) Referência AWS SDK para .NET da API.

DeleteFunction

O código de exemplo a seguir mostra como usar DeleteFunction.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteFunction](#) a Referência AWS SDK para .NET da API.

GetFunction

O código de exemplo a seguir mostra como usar `GetFunction`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [GetFunction](#) na Referência AWS SDK para .NET da API.

Invoke

O código de exemplo a seguir mostra como usar Invoke.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
    }  
}
```

- Consulte detalhes da API em [Invoke](#), na Referência da API AWS SDK para .NET .

ListFunctions

O código de exemplo a seguir mostra como usar ListFunctions.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [ListFunctions](#) na Referência AWS SDK para .NET da API.

UpdateFunctionCode

O código de exemplo a seguir mostra como usar UpdateFunctionCode.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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}.");
}
```

- Para obter detalhes da API, consulte [UpdateFunctionCode](#) na Referência AWS SDK para .NET da API.

UpdateFunctionConfiguration

O código de exemplo a seguir mostra como usar UpdateFunctionConfiguration.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
    }
```

- Para obter detalhes da API, consulte [UpdateFunctionConfiguration](#) na Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Criar uma aplicação para analisar o feedback dos clientes

O exemplo de código a seguir mostra como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

SDK para .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Como transformar dados com o S3 Object Lambda

O exemplo de código a seguir mostra como transformar dados para sua aplicação com o S3 Object Lambda.

SDK para .NET

Mostra como adicionar código personalizado a solicitações GET padrão do S3 para modificar o objeto solicitado e recuperado do S3 e possibilitar que o objeto atenda às necessidades do cliente ou aplicação solicitante.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Lambda

- Amazon S3

Exemplos sem servidor

Como se conectar a um banco de dados do Amazon RDS em uma função do Lambda

O exemplo de código a seguir mostra como implementar uma função do Lambda que se conecte a um banco de dados do RDS. A função faz uma solicitação simples ao banco de dados e exibe o resultado.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando .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" })
        };
    }
}

```

Invocar uma função do Lambda em um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um stream do Kinesis. A função recupera a carga útil do Kinesis, decodifica do Base64 e registra o conteúdo do registro em log.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do Kinesis com o Lambda usando .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}");
            }
            catch { }
        }
    }
}
```

```

        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;
}
}

```

Invocar uma função do Lambda em um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Como consumir um evento do DynamoDB com o Lambda usando .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.");
    }
}
```

Invocar uma função do Lambda de um acionador do Amazon DocumentDB

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo de alterações do DocumentDB. A função recupera a carga útil do DocumentDB e registra em log o conteúdo do registro.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Processando um evento do Amazon DocumentDB com o Lambda ao usar .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 Ns { 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; }
    }
}
```

Invocar uma função do Lambda em um gatinho do Amazon MSK

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um cluster do Amazon MSK. A função recupera a carga útil do MSK e registra em log o conteúdo dos registros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do Amazon MSK com o Lambda usando .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);
            }
        }
    }
}
```

Invocar uma função do Lambda em um acionador do Amazon S3

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo upload de um objeto para um bucket do S3. A função recupera o nome do bucket do S3 e a chave do objeto do parâmetro de evento e chama a API do Amazon S3 para recuperar e registrar em log o tipo de conteúdo do objeto.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do S3 com o Lambda usando .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;
    }
}
}
```

Invocar uma função do Lambda em um acionador do Amazon SNS

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um tópico do SNS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do SNS com o Lambda usando .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;
        }
    }
}
}
```


Invocar uma função do Lambda em um trigger do Amazon SQS

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de uma fila do SQS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do SQS com o Lambda usando .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;
        }
    }
}
```

Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um stream do Kinesis. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Relatar falhas de itens em lote do Kinesis com o Lambda usando o .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; }
    }
}

```

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um fluxo do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando .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;
}
}
```

Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de uma fila do SQS. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Relatar falhas de itens em lote do SQS com o Lambda usando o .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 contribuições da comunidade

Compilar e testar uma aplicação com tecnologia sem servidor

O exemplo de código a seguir mostra como criar e testar uma aplicação com tecnologia sem servidor usando o API Gateway com o Lambda e o DynamoDB.

SDK para .NET

Mostra como compilar e testar uma aplicação com tecnologia sem servidor que consiste em um API Gateway com o Lambda e o DynamoDB usando o SDK .NET.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

MediaConvert exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with MediaConvert.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá MediaConvert

O exemplo de código a seguir mostra como começar a usar o AWS Elemental MediaConvert.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();  
    }  
}  
}
```

- Para obter detalhes da API, consulte [DescribeEndpoints](#) a Referência AWS SDK para .NET da API.

Ações

CreateJob

O código de exemplo a seguir mostra como usar CreateJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Configure o wrapper, o cliente e os locais do arquivo.

```
// 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));

```

Criar o trabalho usando o método wrapper e retorne o ID do trabalho.

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [CreateJob](#) na Referência AWS SDK para .NET da API.

GetJob

O código de exemplo a seguir mostra como usar GetJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Configure o wrapper, o cliente e os locais do arquivo.

```
// 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);
```

Obter um trabalho pelo 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;
}
```

- Para obter detalhes da API, consulte [GetJob](#) a Referência AWS SDK para .NET da API.

ListJobs

O código de exemplo a seguir mostra como usar ListJobs.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Configure o wrapper, o cliente e os locais do arquivo.

```
// 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);

```

Liste os trabalhos usando um status específico.

```

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}.");
});

```

Liste os trabalhos usando um paginador.

```

/// <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;  
    }
```

- Para obter detalhes da API, consulte [ListJobs](#) a Referência AWS SDK para .NET da API.

Exemplos do Amazon MSK usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon MSK.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Exemplos sem servidor](#)

Exemplos sem servidor

Invocar uma função do Lambda em um gatinho do Amazon MSK

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um cluster do Amazon MSK. A função recupera a carga útil do MSK e registra em log o conteúdo dos registros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do Amazon MSK com o Lambda usando .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);
            }
        }
    }
}
```

Exemplos de organizações usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Organizations.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

AttachPolicy

O código de exemplo a seguir mostra como usar `AttachPolicy`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
    }
}
}
```

- Para obter detalhes da API, consulte [AttachPolicy](#) na Referência AWS SDK para .NET da API.

CreateAccount

O código de exemplo a seguir mostra como usar CreateAccount.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}.");
    }
}

```

- Para obter detalhes da API, consulte [CreateAccount](#) na Referência AWS SDK para .NET da API.

CreateOrganization

O código de exemplo a seguir mostra como usar CreateOrganization.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```


- Para obter detalhes da API, consulte [CreateOrganization](#) na Referência AWS SDK para .NET da API.

CreateOrganizationalUnit

O código de exemplo a seguir mostra como usar CreateOrganizationalUnit.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
}
}
```

- Para obter detalhes da API, consulte [CreateOrganizationalUnit](#) na Referência AWS SDK para .NET da API.

CreatePolicy

O código de exemplo a seguir mostra como usar CreatePolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
        }
    }
}

```

- Para obter detalhes da API, consulte [CreatePolicy](#) a Referência AWS SDK para .NET da API.

DeleteOrganization

O código de exemplo a seguir mostra como usar DeleteOrganization.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");  
    }  
}  
}
```

- Para obter detalhes da API, consulte [DeleteOrganization](#) na Referência AWS SDK para .NET da API.

DeleteOrganizationalUnit

O código de exemplo a seguir mostra como usar DeleteOrganizationalUnit.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}.");  
    }  
}
```

- Para obter detalhes da API, consulte [DeleteOrganizationalUnit](#) na Referência AWS SDK para .NET da API.

DeletePolicy

O código de exemplo a seguir mostra como usar DeletePolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}.");
        }
    }
}
```

- Para obter detalhes da API, consulte [DeletePolicy](#) a Referência AWS SDK para .NET da API.

DetachPolicy

O código de exemplo a seguir mostra como usar DetachPolicy.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
        }
    }
}
```

- Para obter detalhes da API, consulte [DetachPolicy](#) a Referência AWS SDK para .NET da API.

ListAccounts

O código de exemplo a seguir mostra como usar ListAccounts.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}

```

- Para obter detalhes da API, consulte [ListAccounts](#) a Referência AWS SDK para .NET da API.

ListOrganizationalUnitsForParent

O código de exemplo a seguir mostra como usar ListOrganizationalUnitsForParent.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}

```

- Para obter detalhes da API, consulte [ListOrganizationalUnitsForParent](#) a Referência AWS SDK para .NET da API.

ListPolicies

O código de exemplo a seguir mostra como usar ListPolicies.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}

```

- Para obter detalhes da API, consulte [ListPolicies](#) na Referência AWS SDK para .NET da API.

Exemplos do Partner Central usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Partner Central.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

CreateOpportunity

O código de exemplo a seguir mostra como usar CreateOpportunity.

SDK para .NET

Crie uma oportunidade.

```
// 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.");
    }
}
}
}

```

- Para obter detalhes da API, consulte [CreateOpportunity](#) a Referência AWS SDK para .NET da API.

GetOpportunity

O código de exemplo a seguir mostra como usar GetOpportunity.

SDK para .NET

Tenha uma oportunidade.

```

// 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.");
    }
}
}
}

```

- Para obter detalhes da API, consulte [GetOpportunity](#) a Referência AWS SDK para .NET da API.

ListOpportunities

O código de exemplo a seguir mostra como usar `ListOpportunities`.

SDK para .NET

Liste oportunidades.

```
// 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.");
}
}
}
}
}

```

- Para obter detalhes da API, consulte [ListOpportunities](#) a Referência AWS SDK para .NET da API.

Exemplos do Amazon Pinpoint usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Pinpoint.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

SendMessage

O código de exemplo a seguir mostra como usar `SendMessage`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Envie uma mensagem de 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;
}
}

```

Envie uma mensagem 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
            }
        }
    };
    SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
    return response;
}
```

- Para obter detalhes da API, consulte [SendMessages](#) na Referência AWS SDK para .NET da API.

Exemplos do Amazon Polly usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Polly.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

DeleteLexicon

O código de exemplo a seguir mostra como usar DeleteLexicon.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}
```

- Para obter detalhes da API, consulte [DeleteLexicon](#) na Referência AWS SDK para .NET da API.

DescribeVoices

O código de exemplo a seguir mostra como usar DescribeVoices.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
}
}

```

- Para obter detalhes da API, consulte [DescribeVoices](#) na Referência AWS SDK para .NET da API.

GetLexicon

O código de exemplo a seguir mostra como usar GetLexicon.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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);
        }
    }
}
```

- Para obter detalhes da API, consulte [GetLexicon](#) na Referência AWS SDK para .NET da API.

ListLexicons

O código de exemplo a seguir mostra como usar ListLexicons.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"Name: {lexicon.Name}");
                    Console.WriteLine($"  Alphabet: {attributes.Alphabet}");
                    Console.WriteLine($"  LanguageCode:
{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}");
}
}
}

```

- Para obter detalhes da API, consulte [ListLexicons](#) na Referência AWS SDK para .NET da API.

PutLexicon

O código de exemplo a seguir mostra como usar PutLexicon.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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);
        }
    }
}

```

- Para obter detalhes da API, consulte [PutLexicon](#) na Referência AWS SDK para .NET da API.

SynthesizeSpeech

O código de exemplo a seguir mostra como usar SynthesizeSpeech.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
    }
}

```

Sintetize fala a partir de texto usando marcas de fala com o Amazon Polly usando um SDK. AWS

```
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}");  
}  
}  
}
```

- Para obter detalhes da API, consulte [SynthesizeSpeech](#) a Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação para analisar o feedback dos clientes

O exemplo de código a seguir mostra como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

SDK para .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemplos do Amazon RDS usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon RDS.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)
- [Exemplos sem servidor](#)

Conceitos básicos

Olá, Amazon RDS

O exemplo de código a seguir mostra como começar a usar o Amazon RDS.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
        }
    }
}
```

- Para obter detalhes da API, consulte [Descrver DBInstances](#) na Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um grupo de parâmetros de banco de dados e definir os valores dos parâmetros.
- Criar uma instância de banco de dados configurada para usar o grupo de parâmetros. A instância de banco de dados também contém um banco de dados.
- Criar um snapshot da instância.
- Exclua a instância e o grupo de parâmetros.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
/// <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<RDSEntity>()
    )
    .Build();

logger = LoggerFactory.Create(builder =>
{
    builder.AddConsole();
}).CreateLogger<RDSInstanceScenario>();

rdsWrapper = host.Services.GetRequiredService<RDSEntity>();

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}. 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(
            $"\\n\\tParameter: {p.ParameterName}." +
            $"\\n\\tDescription: {p.Description}." +
            $"\\n\\tAllowed Values: {p.AllowedValues}." +
            $"\\n\\tValue: {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(
                $"{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(
        $"\\n{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.DBInstanceIdIdentifier}.");
    await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdIdentifier);

    // 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.DBInstanceIdIdentifier !=
newInstance.DBInstanceIdIdentifier);
        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);
}

```

Os métodos de wrapper usados pelo cenário para as ações de instância de banco de dados.

```

/// <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="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;
    }

```

```

    }

    /// <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;
    }

```

```

/// <summary>
/// Delete a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdIdentifier">DB instance identifier.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdIdentifier = dbInstanceIdIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}

```

Os métodos de wrapper usados pelo cenário para o grupos de parâmetros de banco de dados.

```

/// <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;
}

```

Os métodos de wrapper usados pelo cenário para as ações de snapshot de banco de dados.

```

/// <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="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;
    }
}

```

```
/// <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;
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CriarDBInstance](#)
 - [Criar DBParameter grupo](#)
 - [CriarDBSnapshot](#)
 - [ExcluirDBInstance](#)
 - [Excluir DBParameter grupo](#)
 - [Descreva DBEngine as versões](#)
 - [DescreverDBInstances](#)
 - [Descreva DBParameter os grupos](#)
 - [DescreverDBParameters](#)
 - [DescreverDBSnapshots](#)
 - [DescribeOrderableDBInstanceOpções](#)

- [Modificar DBParameter grupo](#)

Ações

CreateDBInstance

O código de exemplo a seguir mostra como usar CreateDBInstance.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Criar DBInstance](#) na referência AWS SDK para .NET da API.

CreateDBParameterGroup

O código de exemplo a seguir mostra como usar CreateDBParameterGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [Criar DBParameter grupo](#) na referência AWS SDK para .NET da API.

CreateDBSnapshot

O código de exemplo a seguir mostra como usar CreateDBSnapshot.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <summary>
/// Create a snapshot of a DB instance.
/// </summary>
/// <param name="dbInstanceId">DB instance identifier.</param>
/// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
/// <returns>DB snapshot object.</returns>
public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceId,
    string snapshotIdentifier)

```

```
{
    var response = await _amazonRDS.CreateDBSnapshotAsync(
        new CreateDBSnapshotRequest()
        {
            DBSnapshotIdentifier = snapshotIdentifier,
            DBInstanceIdentifier = dbInstanceIdentifier
        });

    return response.DBSnapshot;
}
```

- Para obter detalhes da API, consulte [Criar DBSnapshot](#) na referência AWS SDK para .NET da API.

DeleteDBInstance

O código de exemplo a seguir mostra como usar DeleteDBInstance.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Excluir DBInstance](#) na Referência AWS SDK para .NET da API.

DeleteDBParameterGroup

O código de exemplo a seguir mostra como usar DeleteDBParameterGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Excluir DBParameter grupo](#) na Referência AWS SDK para .NET da API.

DescribeDBEngineVersions

O código de exemplo a seguir mostra como usar DescribeDBEngineVersions.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Descrever DBEngine as versões](#) na Referência AWS SDK para .NET da API.

DescribeDBInstances

O código de exemplo a seguir mostra como usar DescribeDBInstances.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Descrever DBInstances](#) na Referência AWS SDK para .NET da API.

DescribeDBParameterGroups

O código de exemplo a seguir mostra como usar `DescribeDBParameterGroups`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Descrever DBParameter grupos](#) na referência AWS SDK para .NET da API.

DescribeDBParameters

O código de exemplo a seguir mostra como usar `DescribeDBParameters`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Descrever DBParameters](#) na Referência AWS SDK para .NET da API.

DescribeDBSnapshots

O código de exemplo a seguir mostra como usar DescribeDBSnapshots.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Descrever DBSnapshots](#) na Referência AWS SDK para .NET da API.

DescribeOrderableDBInstanceOptions

O código de exemplo a seguir mostra como usar `DescribeOrderableDBInstanceOptions`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeOrderableDBInstanceOpções](#) na Referência AWS SDK para .NET da API.

ModifyDBParameterGroup

O código de exemplo a seguir mostra como usar ModifyDBParameterGroup.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [Modificar DBParameter grupo](#) na Referência AWS SDK para .NET da API.

Cenários

Crie um rastreador de itens de trabalho do Aurora Sem Servidor

O exemplo de código a seguir mostra como criar uma aplicação web que rastreia os itens de trabalho em um banco de dados do Amazon Aurora Sem Servidor e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar o AWS SDK para .NET para criar um aplicativo web que rastreia itens de trabalho em um banco de dados Amazon Aurora e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa um front-end criado com o React.js para interagir com um back-end RESTful do .NET.

- Integre um aplicativo web React com AWS serviços.
- Liste, adicione, atualize e exclua itens em uma tabela do Aurora.
- Envie um relatório por e-mail dos itens de trabalho filtrados usando o Amazon SES.
- Implante e gerencie recursos de exemplo com o AWS CloudFormation script incluído.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Aurora
- Amazon RDS
- Serviços de dados do Amazon RDS
- Amazon SES

Exemplos sem servidor

Como se conectar a um banco de dados do Amazon RDS em uma função do Lambda

O exemplo de código a seguir mostra como implementar uma função do Lambda que se conecte a um banco de dados do RDS. A função faz uma solicitação simples ao banco de dados e exibe o resultado.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando .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" });
};
}
}
```

Exemplos de serviços de dados do Amazon RDS usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET Amazon RDS Data Service.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Cenários](#)

Cenários

Crie um rastreador de itens de trabalho do Aurora Sem Servidor

O exemplo de código a seguir mostra como criar uma aplicação web que rastreia os itens de trabalho em um banco de dados do Amazon Aurora Sem Servidor e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar o AWS SDK para .NET para criar um aplicativo web que rastreia itens de trabalho em um banco de dados Amazon Aurora e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa um front-end criado com o React.js para interagir com um back-end RESTful do .NET.

- Integre um aplicativo web React com AWS serviços.
- Liste, adicione, atualize e exclua itens em uma tabela do Aurora.
- Envie um relatório por e-mail dos itens de trabalho filtrados usando o Amazon SES.
- Implante e gerencie recursos de exemplo com o AWS CloudFormation script incluído.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Aurora
- Amazon RDS
- Serviços de dados do Amazon RDS
- Amazon SES

Exemplos do Amazon Rekognition usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Rekognition.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

CompareFaces

O código de exemplo a seguir mostra como usar CompareFaces.

Para obter mais informações, consulte [Comparação de faces em imagens](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
    }
}
```

- Para obter detalhes da API, consulte [CompareFaces](#) a Referência AWS SDK para .NET da API.

CreateCollection

O código de exemplo a seguir mostra como usar CreateCollection.

Para obter mais informações, consulte [Criar uma coleção](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [CreateCollection](#) na Referência AWS SDK para .NET da API.

DeleteCollection

O código de exemplo a seguir mostra como usar DeleteCollection.

Para obter mais informações, consulte [Excluir uma coleção](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [DeleteCollection](#) na Referência AWS SDK para .NET da API.

DeleteFaces

O código de exemplo a seguir mostra como usar DeleteFaces.

Para obter mais informações, consulte [Excluir faces de uma coleção](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");  
    });  
}  
}
```

- Para obter detalhes da API, consulte [DeleteFaces](#) a Referência AWS SDK para .NET da API.

DescribeCollection

O código de exemplo a seguir mostra como usar `DescribeCollection`.

Para obter mais informações, consulte [Descrever uma coleção](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [DescribeCollection](#) na Referência AWS SDK para .NET da API.

DetectFaces

O código de exemplo a seguir mostra como usar DetectFaces.

Para obter mais informações, consulte [Detectar faces em uma imagem](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}
}

```

Exiba as informações da caixa delimitadora de todas as faces em uma imagem.

```

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}");
}
}

```

- Para obter detalhes da API, consulte [DetectFaces](#) a Referência AWS SDK para .NET da API.

DetectLabels

O código de exemplo a seguir mostra como usar DetectLabels.

Para obter mais informações, consulte [Detectar rótulos em uma imagem](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}
}

```

Detecte rótulos em um arquivo de imagem armazenado em seu computador.

```

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);
    }
}
}

```

- Para obter detalhes da API, consulte [DetectLabels](#) a Referência AWS SDK para .NET da API.

DetectModerationLabels

O código de exemplo a seguir mostra como usar DetectModerationLabels.

Para obter mais informações, consulte [Detectar imagens impróprias](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
}
}
```

- Para obter detalhes da API, consulte [DetectModerationLabels](#) na Referência AWS SDK para .NET da API.

DetectText

O código de exemplo a seguir mostra como usar DetectText.

Para obter mais informações, consulte [Detectar texto em uma imagem](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
        }
    }
}
```

```
    }  
  }  
}
```

- Para obter detalhes da API, consulte [DetectText](#) Referência AWS SDK para .NET da API.

GetCelebrityInfo

O código de exemplo a seguir mostra como usar GetCelebrityInfo.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
        });
    }
}
```

- Para obter detalhes da API, consulte [GetCelebrityInfo](#) a Referência AWS SDK para .NET da API.

IndexFaces

O código de exemplo a seguir mostra como usar IndexFaces.

Para obter mais informações, consulte [Adicionar faces a uma coleção](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}
```

- Para obter detalhes da API, consulte [IndexFaces](#) a Referência AWS SDK para .NET da API.

ListCollections

O código de exemplo a seguir mostra como usar ListCollections.

Para obter mais informações, consulte [Listar coleções](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}
```

- Para obter detalhes da API, consulte [ListCollections](#) na Referência AWS SDK para .NET da API.

ListFaces

O código de exemplo a seguir mostra como usar ListFaces.

Para obter mais informações, consulte [Listar faces em uma coleção](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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));
    }
}

```

- Para obter detalhes da API, consulte [ListFaces](#) a Referência AWS SDK para .NET da API.

RecognizeCelebrities

O código de exemplo a seguir mostra como usar `RecognizeCelebrities`.

Para obter mais informações, consulte [Reconhecer celebridades em uma imagem](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.UrlsWithEach(url =>
        {
            Console.WriteLine(url);
        });
    });

    Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
    }
}

```

- Para obter detalhes da API, consulte [RecognizeCelebrities](#) a Referência AWS SDK para .NET da API.

SearchFaces

O código de exemplo a seguir mostra como usar SearchFaces.

Para obter mais informações, consulte [Pesquisar uma face \(face ID\)](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");  
        }  
    }  
}
```

- Para obter detalhes da API, consulte [SearchFaces](#) a Referência AWS SDK para .NET da API.

SearchFacesByImage

O código de exemplo a seguir mostra como usar SearchFacesByImage.

Para obter mais informações, consulte [Pesquisar uma face \(imagem\)](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
});
}
}

```

- Para obter detalhes da API, consulte [SearchFacesByImage](#) a Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Detectar objetos em imagens

O exemplo de código a seguir mostra como construir uma aplicação que usa o Amazon Rekognition para detectar objetos por categoria em imagens.

SDK para .NET

Mostra como usar a API .NET do Amazon Rekognition para construir uma aplicação que usa o Amazon Rekognition para identificar objetos por categoria em imagens localizadas em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemplos de registro de domínio do Route 53 usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o registro de domínio AWS SDK para .NET com o Route 53.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Registro de domínios do Olá, Route 53

O exemplo de código a seguir mostra como começar a usar o registro de domínio do Route 53.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}
```

- Para obter detalhes da API, consulte [ListPrices](#) a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Listar os domínios atuais e as operações do ano passado.
- Ver o faturamento do ano passado e os preços dos tipos de domínio.
- Receber sugestões de domínio.
- Verificar a disponibilidade e a transferibilidade de um domínio.
- Opcionalmente, solicitar o registro de um domínio.
- Obter os detalhes de uma operação.
- Opcionalmente, obtenha os detalhes de um domínio.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
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($"\\tTransferability: {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($"\\tNote: This example uses domain request settings in
settings.json.");
        Console.WriteLine($"\\tTo change the domain registration settings, set the
values in that file.");
        Console.WriteLine($"\\tRemember, registering an actual domain will incur an
account billing cost.");
        Console.WriteLine($"\\tWould 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(
                $"{Environment.NewLine}Registration 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;
    }
}

```

Os métodos de wrapper usados pelo cenário para as ações do registro de domínios do 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.Value;
    }

```

```

    }
    );
    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.";
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CheckDomainAvailability](#)
 - [CheckDomainTransferability](#)
 - [GetDomainDetail](#)
 - [GetDomainSuggestions](#)
 - [GetOperationDetail](#)
 - [ListDomains](#)
 - [ListOperations](#)
 - [ListPrices](#)
 - [RegisterDomain](#)
 - [ViewBilling](#)

Ações

CheckDomainAvailability

O código de exemplo a seguir mostra como usar CheckDomainAvailability.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CheckDomainAvailability](#) a Referência AWS SDK para .NET da API.

CheckDomainTransferability

O código de exemplo a seguir mostra como usar CheckDomainTransferability.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CheckDomainTransferability](#) a Referência AWS SDK para .NET da API.

GetDomainDetail

O código de exemplo a seguir mostra como usar GetDomainDetail.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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.";
        }
    }

```

- Para obter detalhes da API, consulte [GetDomainDetail](#) a Referência AWS SDK para .NET da API.

GetDomainSuggestions

O código de exemplo a seguir mostra como usar GetDomainSuggestions.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetDomainSuggestions](#) na Referência AWS SDK para .NET da API.

GetOperationDetail

O código de exemplo a seguir mostra como usar `GetOperationDetail`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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}.";
    }
}
```

- Para obter detalhes da API, consulte [GetOperationDetail](#) a Referência AWS SDK para .NET da API.

ListDomains

O código de exemplo a seguir mostra como usar `ListDomains`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListDomains](#) a Referência AWS SDK para .NET da API.

ListOperations

O código de exemplo a seguir mostra como usar `ListOperations`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListOperations](#) na Referência AWS SDK para .NET da API.

ListPrices

O código de exemplo a seguir mostra como usar `ListPrices`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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();
}
```

- Para obter detalhes da API, consulte [ListPrices](#) a Referência AWS SDK para .NET da API.

RegisterDomain

O código de exemplo a seguir mostra como usar RegisterDomain.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).


```

    /// <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;
        }
    }
}

```

- Para obter detalhes da API, consulte [RegisterDomain](#) na Referência AWS SDK para .NET da API.

ViewBilling

O código de exemplo a seguir mostra como usar ViewBilling.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ViewBilling](#) Referência AWS SDK para .NET da API.

Exemplos do Amazon S3 usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon S3.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)
- [Exemplos sem servidor](#)

Ações

CopyObject

O código de exemplo a seguir mostra como usar CopyObject.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Copie um objeto usando uma solicitação condicional.

```
/// <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;
}
}

```

- Para obter detalhes da API, consulte [CopyObject](#) a Referência AWS SDK para .NET da API.

CreateBucket

O código de exemplo a seguir mostra como usar CreateBucket.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie um bucket com o bloqueio de objetos habilitado.

```
/// <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;
    }
}
```

- Para obter detalhes da API, consulte [CreateBucket](#) na Referência AWS SDK para .NET da API.

DeleteBucketCors

O código de exemplo a seguir mostra como usar DeleteBucketCors.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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);
}
```

- Para obter detalhes da API, consulte [DeleteBucketCors](#) na Referência AWS SDK para .NET da API.

DeleteBucketLifecycle

O código de exemplo a seguir mostra como usar DeleteBucketLifecycle.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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);
}
```

- Para obter detalhes da API, consulte [DeleteBucketLifecycle](#) a Referência AWS SDK para .NET da API.

DeleteObject

O código de exemplo a seguir mostra como usar DeleteObject.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Exclua um objeto de um bucket do S3 sem versionamento.

```
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.USSouth1.
        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.");
        }
    }
}

```

Exclua um objeto de um bucket do S3 com versionamento.

```

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 co 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;
    }
}

```

- Para obter detalhes da API, consulte [DeleteObject](#) a Referência AWS SDK para .NET da API.

DeleteObjects

O código de exemplo a seguir mostra como usar DeleteObjects.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Exclua vários objetos de um bucket do S3 sem versionamento.

```
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;
}
}

```

Exclua vários objetos de um bucket do S3 com versionamento.

```

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;
    }
}

```

- Para obter detalhes da API, consulte [DeleteObjects](#) na Referência AWS SDK para .NET da API.

GetBucketAcl

O código de exemplo a seguir mostra como usar `GetBucketAcl`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetBucketAcl](#) Referência AWS SDK para .NET da API.

GetBucketCors

O código de exemplo a seguir mostra como usar `GetBucketCors`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetBucketCors](#) a Referência AWS SDK para .NET da API.

GetBucketEncryption

O código de exemplo a seguir mostra como usar `GetBucketEncryption`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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(
                $"Algorithm:
{encryptionSettings.ServerSideEncryptionByDefault.ServerSideEncryptionAlgorithm}");
            Console.WriteLine(
                $"Key:
{encryptionSettings.ServerSideEncryptionByDefault.ServerSideEncryptionKeyManagementServiceKey
            }
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine(ex.ErrorCode == "InvalidBucketName"
                ? $"Bucket {bucketName} was not found."
                : $"Unable to get bucket encryption for bucket {bucketName},
{ex.Message}");
        }
    }
}

```

```
}
```

- Para obter detalhes da API, consulte [GetBucketEncryption](#) na Referência AWS SDK para .NET da API.

GetBucketLifecycleConfiguration

O código de exemplo a seguir mostra como usar `GetBucketLifecycleConfiguration`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetBucketLifecycleConfiguration](#) na Referência AWS SDK para .NET da API.

GetBucketWebsite

O código de exemplo a seguir mostra como usar `GetBucketWebsite`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
// 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}");
```

- Para obter detalhes da API, consulte [GetBucketWebsite](#) na Referência AWS SDK para .NET da API.

GetObject

O código de exemplo a seguir mostra como usar `GetObject`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Obtenha um objeto usando uma solicitação condicional.

```
/// <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;
}
}

```

- Para obter detalhes da API, consulte [GetObject](#) Referência AWS SDK para .NET da API.

GetObjectLegalHold

O código de exemplo a seguir mostra como usar `GetObjectLegalHold`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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}:

" +
            $"Status: {response.LegalHold.Status}");
        return response.LegalHold;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Unable to fetch legal hold: '{ex.Message}'");
        return new ObjectLockLegalHold();
    }
}
```

- Para obter detalhes da API, consulte [GetObjectLegalHold](#) da Referência AWS SDK para .NET da API.

GetObjectLockConfiguration

O código de exemplo a seguir mostra como usar `GetObjectLockConfiguration`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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($"Unable to fetch object lock config:
'{ex.Message}');
        return new ObjectLockConfiguration();
    }
}
```

- Para obter detalhes da API, consulte [GetObjectLockConfiguration](#) Referência AWS SDK para .NET da API.

GetObjectRetention

O código de exemplo a seguir mostra como usar `GetObjectRetention`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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();
    }
}

```

- Para obter detalhes da API, consulte [GetObjectRetention](#) a Referência AWS SDK para .NET da API.

ListBuckets

O código de exemplo a seguir mostra como usar ListBuckets.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

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);
    }
}

```

- Para obter detalhes da API, consulte [ListBuckets](#) na Referência AWS SDK para .NET da API.

ListObjectVersions

O código de exemplo a seguir mostra como usar ListObjectVersions.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}'");  
    }  
}  
}
```

- Para obter detalhes da API, consulte [ListObjectVersions](#) na Referência AWS SDK para .NET da API.

ListObjectsV2

O código de exemplo a seguir mostra como usar ListObjectsV2.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Liste objetos com um paginador.

```
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}");
        }
    }
}
}

```

- Para obter detalhes da API, consulte [ListObjectsV2](#) na Referência AWS SDK para .NET da API.

PutBucketAccelerateConfiguration

O código de exemplo a seguir mostra como usar PutBucketAccelerateConfiguration.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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");
    }
}
}
```

- Para obter detalhes da API, consulte [PutBucketAccelerateConfiguration](#) na Referência AWS SDK para .NET da API.

PutBucketAcl

O código de exemplo a seguir mostra como usar PutBucketAcl.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;  
}
```

- Para obter detalhes da API, consulte [PutBucketAcl](#) Referência AWS SDK para .NET da API.

PutBucketCors

O código de exemplo a seguir mostra como usar PutBucketCors.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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);  
}
```

- Para obter detalhes da API, consulte [PutBucketCors](#) Referência AWS SDK para .NET da API.

PutBucketEncryption

O código de exemplo a seguir mostra como usar PutBucketEncryption.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [PutBucketEncryption](#) na Referência AWS SDK para .NET da API.

PutBucketLifecycleConfiguration

O código de exemplo a seguir mostra como usar PutBucketLifecycleConfiguration.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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);
}
```

- Para obter detalhes da API, consulte [PutBucketLifecycleConfiguration](#) na Referência AWS SDK para .NET da API.

PutBucketLogging

O código de exemplo a seguir mostra como usar PutBucketLogging.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
    }
}
```

- Para obter detalhes da API, consulte [PutBucketLogging](#) na Referência AWS SDK para .NET da API.

PutBucketNotificationConfiguration

O código de exemplo a seguir mostra como usar PutBucketNotificationConfiguration.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
}

```

- Para obter detalhes da API, consulte [PutBucketNotificationConfiguration](#) na Referência AWS SDK para .NET da API.

PutBucketWebsite

O código de exemplo a seguir mostra como usar PutBucketWebsite.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
// Put the website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
{
    BucketName = bucketName,
    WebsiteConfiguration = new WebsiteConfiguration()
    {
        IndexDocumentSuffix = indexDocumentSuffix,
        ErrorDocument = errorDocument,
    },
};
PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- Para obter detalhes da API, consulte [PutBucketWebsite](#) a Referência AWS SDK para .NET da API.

PutObject

O código de exemplo a seguir mostra como usar `PutObject`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Faça upload de um objeto com criptografia do lado do servidor.

```

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.USSouth1.
        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");
    }
}
}

```

Coloque um objeto usando uma solicitação condicional.

```

/// <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;
    }
}
```

- Para obter detalhes da API, consulte [PutObject](#) Referência AWS SDK para .NET da API.

PutObjectLegalHold

O código de exemplo a seguir mostra como usar PutObjectLegalHold.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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($"{Environment.NewLine}Modified legal hold for {objectKey} in
{bucketName}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"{Environment.NewLine}Error modifying legal hold: '{ex.Message}'");
        return false;
    }
}
```


- Para obter detalhes da API, consulte [PutObjectLegalHold](#) da Referência AWS SDK para .NET da API.

PutObjectLockConfiguration

O código de exemplo a seguir mostra como usar PutObjectLockConfiguration.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Defina a configuração de Bloqueio de Objetos de um bucket.

```
/// <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;
}
}

```

Defina o período de retenção padrão de um bucket.

```

/// <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;
}
}

```

- Para obter detalhes da API, consulte [PutObjectLockConfiguration](#) Referência AWS SDK para .NET da API.

PutObjectRetention

O código de exemplo a seguir mostra como usar PutObjectRetention.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
}
```

- Para obter detalhes da API, consulte [PutObjectRetention](#) na Referência AWS SDK para .NET da API.

RestoreObject

O código de exemplo a seguir mostra como usar `RestoreObject`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.USWest2;

        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}");
}
}
```

- Para obter detalhes da API, consulte [RestoreObject](#) Referência AWS SDK para .NET da API.

Cenários

Criar um URL pré-assinado

O exemplo de código a seguir mostra como criar um URL pré-assinado para o Amazon S3 e fazer upload de um objeto.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Gere um URL pré-assinado que possa executar uma ação do Amazon S3 por tempo limitado.

```

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;
    }
}

```

Gere um URL pré-assinado e faça um upload usando esse 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.");
    }
}

in
/// <summary>
/// Uploads an object to an Amazon S3 bucket using the presigned URL passed
/// 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;
}

call
/// <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
param>
/// 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;
}
}
```

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Detectar objetos em imagens

O exemplo de código a seguir mostra como construir uma aplicação que usa o Amazon Rekognition para detectar objetos por categoria em imagens.

SDK para .NET

Mostra como usar a API .NET do Amazon Rekognition para construir uma aplicação que usa o Amazon Rekognition para identificar objetos por categoria em imagens localizadas em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Conceitos básicos de criptografia

O exemplo de código a seguir mostra como começar a usar a criptografia de objetos do Amazon S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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>

```

```

    call    /// <param name="client">The initialized Amazon S3 client object used to
    object  /// GetObjectAsync.</param>
    param>  /// <param name="bucketName">The name of the Amazon S3 bucket where the
    object  /// is located.</param>
    param>  /// <param name="keyName">The name of the Amazon S3 object to download.</
    param>  /// <param name="base64Key">The encryption key used to encrypt the
    param>  /// object.</param>
    param>  /// <param name="putObjectRequest">The PutObjectRequest used to upload
    param>  /// 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);
}
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CopyObject](#)
 - [GetObject](#)
 - [GetObjectMetadata](#)

Conceitos básicos de etiquetas

O exemplo de código a seguir mostra como começar a usar etiquetas de objetos do Amazon S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}'");
    }
}
}

```

- Para obter detalhes da API, consulte [GetObjectTagging](#) Referência AWS SDK para .NET da API.

Bloquear objetos do Amazon S3

O exemplo de código a seguir mostra como trabalhar com os recursos de bloqueio de objetos do S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo demonstrando os recursos de bloqueio de objetos do 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 =
                            $"Would 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;
    }
}

```

```
}
```

Uma classe de wrapper para funções do 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($"\\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();
    }
}

/// <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}:
" +
                $"\\n\\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\\tUnable 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($"\\tBucket 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($"\\tUnable 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($"Delete for {bucketName} complete.");
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Unable to delete bucket {bucketName}: " +
ex.Message);
        return false;
    }
}
}

```

```
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetention](#)
 - [PutObjectLegalHold](#)
 - [PutObjectLockConfiguration](#)
 - [PutObjectRetention](#)

Fazer solicitações condicionais

O exemplo de código a seguir mostra como adicionar pré-condições a solicitações do Amazon S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo que demonstre os recursos de solicitação condicional do 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;
    }
}

```

Uma classe de wrapper para funções do 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;
    }
}
}
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CopyObject](#)
 - [GetObject](#)

- [PutObject](#)

Gerenciar listas de controle de acesso (ACLs)

O exemplo de código a seguir mostra como gerenciar listas de controle de acesso (ACLs) para buckets do Amazon S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.AccessControllList;
    }

    /// <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,
        });

        S3AccessControllList acl = aclResponse.AccessControllList;

        // 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,
    });
}

}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .

- [GetBucketAcl](#)

- [GetObjectAcl](#)
- [PutBucketAcl](#)
- [PutObjectAcl](#)

Executar uma cópia multipart

O exemplo de código a seguir mostra como executar uma cópia multipart de um objeto do Amazon S3.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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");
    }

```

```
}  
}  
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CompleteMultipartUpload](#)
 - [CreateMultipartUpload](#)
 - [GetObjectMetadata](#)
 - [UploadPartCopy](#)

Como transformar dados com o S3 Object Lambda

O exemplo de código a seguir mostra como transformar dados para sua aplicação com o S3 Object Lambda.

SDK para .NET

Mostra como adicionar código personalizado a solicitações GET padrão do S3 para modificar o objeto solicitado e recuperado do S3 e possibilitar que o objeto atenda às necessidades do cliente ou aplicação solicitante.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Lambda
- Amazon S3

Fazer upload ou download de arquivos grandes

O exemplo de código a seguir mostra como fazer upload ou download de arquivos grandes de e para o Amazon S3.

Para obter mais informações, consulte [Carregar um objeto usando carregamento fracionado](#).

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Chame funções que transferem arquivos de e para um bucket do S3 usando o Amazon TransferUtility S3.

```
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);
    }
}
```

Carregue um único arquivo.

```

/// <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;
        }
    }
}

```

Carregue um diretório local inteiro.

```

/// <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 { }
    }
}

```

```

        }
        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;
    }
}

```

Baixe um único arquivo.

```

/// <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}"));
}
```

Baixe o conteúdo de um bucket do 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 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;
}

```

Acompanhe o progresso de um upload usando TransferUtility o.

```

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}");
}
}

```

Faça upload de um objeto com criptografia.

```

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);
}
}
}

```

Exemplos sem servidor

Invocar uma função do Lambda em um acionador do Amazon S3

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo upload de um objeto para um bucket do S3. A função recupera o nome do bucket do S3 e a chave do objeto do parâmetro de evento e chama a API do Amazon S3 para recuperar e registrar em log o tipo de conteúdo do objeto.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do S3 com o Lambda usando .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 Exemplos de IA usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com SageMaker IA.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)

Conceitos básicos

Olá SageMaker AI

O exemplo de código a seguir mostra como começar a usar a SageMaker IA.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"\\tInstance:
{notebookInstance.NotebookInstanceName}");
            Console.WriteLine($"\\tArn: {notebookInstance.NotebookInstanceArn}");
            Console.WriteLine($"\\tCreation Date:
{notebookInstance.CreationTime.ToShortDateString()}");
            Console.WriteLine();
        }
    }
}

```

- Para obter detalhes da API, consulte [ListNotebookInstances](#) a Referência AWS SDK para .NET da API.

Ações

CreatePipeline

O código de exemplo a seguir mostra como usar CreatePipeline.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
}

```

- Para obter detalhes da API, consulte [CreatePipeline](#) na Referência AWS SDK para .NET da API.

DeletePipeline

O código de exemplo a seguir mostra como usar DeletePipeline.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [DeletePipeline](#) na Referência AWS SDK para .NET da API.

DescribePipelineExecution

O código de exemplo a seguir mostra como usar `DescribePipelineExecution`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribePipelineExecution](#) na Referência AWS SDK para .NET da API.

StartPipelineExecution

O código de exemplo a seguir mostra como usar `StartPipelineExecution`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [StartPipelineExecution](#) na Referência AWS SDK para .NET da API.

UpdatePipeline

O código de exemplo a seguir mostra como usar UpdatePipeline.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
    }  
}
```

- Para obter detalhes da API, consulte [UpdatePipeline](#) a Referência AWS SDK para .NET da API.

Cenários

Conceitos básicos de trabalhos geoespaciais e pipelines

O exemplo de código a seguir mostra como:

- Configurar recursos para um pipeline.
- Configurar um pipeline que executa um trabalho geoespacial.
- Iniciar a execução de um pipeline.
- Monitorar o status da execução.
- Ver a saída do pipeline.
- Limpar recursos.

Para obter mais informações, consulte [Criar e executar SageMaker pipelines usando o AWS SDKs Community.aws](#).

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie uma classe que envolva as operações de 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) }
        }
    });
#pragma warning restore SageMaker1002
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;
    }
}

```

Crie uma função que gerencie retornos de chamada do pipeline de 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.DefaultLambdaJsonSeriali

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.");
        }
    }
}
}
}

```

Execute um cenário interativo em um prompt de comando.

```
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.USWest2 };
        // 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 = "{" +
        "\\nVersion\\": \\\"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.USSouthWest2
        });

        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($"Pipeline 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 = lambdaRoleName
        });
        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;  
}  
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Exemplos de Secrets Manager usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Secrets Manager.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

GetSecretValue

O código de exemplo a seguir mostra como usar `GetSecretValue`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
    }
}
}

```

- Para obter detalhes da API, consulte [GetSecretValue](#) a Referência AWS SDK para .NET da API.

Exemplos do Amazon SES usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon SES.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

CreateTemplate

O código de exemplo a seguir mostra como usar CreateTemplate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateTemplate](#) a Referência AWS SDK para .NET da API.

DeleteIdentity

O código de exemplo a seguir mostra como usar DeleteIdentity.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteIdentity](#) a Referência AWS SDK para .NET da API.

DeleteTemplate

O código de exemplo a seguir mostra como usar DeleteTemplate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteTemplate](#) a Referência AWS SDK para .NET da API.

GetIdentityVerificationAttributes

O código de exemplo a seguir mostra como usar `GetIdentityVerificationAttributes`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [GetIdentityVerificationAttributes](#) a Referência AWS SDK para .NET da API.

GetSendQuota

O código de exemplo a seguir mostra como usar GetSendQuota.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

```
}
```

- Para obter detalhes da API, consulte [GetSendQuota](#) a Referência AWS SDK para .NET da API.

ListIdentities

O código de exemplo a seguir mostra como usar `ListIdentities`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [ListIdentities](#) a Referência AWS SDK para .NET da API.

ListTemplates

O código de exemplo a seguir mostra como usar ListTemplates.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListTemplates](#) a Referência AWS SDK para .NET da API.

SendEmail

O código de exemplo a seguir mostra como usar SendEmail.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [SendEmail](#) a Referência AWS SDK para .NET da API.

SendTemplatedEmail

O código de exemplo a seguir mostra como usar SendTemplatedEmail.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [SendTemplatedEmail](#) Referência AWS SDK para .NET da API.

VerifyEmailIdentity

O código de exemplo a seguir mostra como usar `VerifyEmailIdentity`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [VerifyEmailIdentity](#) a Referência AWS SDK para .NET da API.

Cenários

Criar uma aplicação Web para monitorar dados do DynamoDB

O exemplo de código a seguir mostra como criar uma aplicação Web que monitora itens de trabalho em uma tabela do Amazon DynamoDB e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar a API .NET do Amazon DynamoDB para construir uma aplicação Web dinâmica que monitora os dados de trabalho do DynamoDB.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- DynamoDB
- Amazon SES

Crie um rastreador de itens de trabalho do Aurora Sem Servidor

O exemplo de código a seguir mostra como criar uma aplicação web que rastreia os itens de trabalho em um banco de dados do Amazon Aurora Sem Servidor e usa o Amazon Simple Email Service (Amazon SES) para enviar relatórios.

SDK para .NET

Mostra como usar o AWS SDK para .NET para criar um aplicativo web que rastreia itens de trabalho em um banco de dados Amazon Aurora e envia relatórios por e-mail usando o Amazon Simple Email Service (Amazon SES). Este exemplo usa um front-end criado com o React.js para interagir com um back-end RESTful do .NET.

- Integre um aplicativo web React com AWS serviços.
- Liste, adicione, atualize e exclua itens em uma tabela do Aurora.
- Envie um relatório por e-mail dos itens de trabalho filtrados usando o Amazon SES.
- Implante e gerencie recursos de exemplo com o AWS CloudFormation script incluído.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Aurora
- Amazon RDS
- Serviços de dados do Amazon RDS
- Amazon SES

Detectar objetos em imagens

O exemplo de código a seguir mostra como construir uma aplicação que usa o Amazon Rekognition para detectar objetos por categoria em imagens.

SDK para .NET

Mostra como usar a API .NET do Amazon Rekognition para construir uma aplicação que usa o Amazon Rekognition para identificar objetos por categoria em imagens localizadas em um bucket do Amazon Simple Storage Service (Amazon S3). A aplicação envia uma notificação por e-mail ao administrador com os resultados usando o Amazon Simple Email Service (Amazon SES).

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Exemplos da API v2 do Amazon SES usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando a AWS SDK para .NET API v2 do Amazon SES.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

CreateContact

O código de exemplo a seguir mostra como usar CreateContact.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateContacta](#) Referência AWS SDK para .NET da API.

CreateContactList

O código de exemplo a seguir mostra como usar CreateContactList.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateContactList](#) na Referência AWS SDK para .NET da API.

CreateEmailIdentity

O código de exemplo a seguir mostra como usar `CreateEmailIdentity`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
}
```

- Para obter detalhes da API, consulte [CreateEmailIdentity](#) a Referência AWS SDK para .NET da API.

CreateEmailTemplate

O código de exemplo a seguir mostra como usar CreateEmailTemplate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [CreateEmailTemplate](#) a Referência AWS SDK para .NET da API.

DeleteContactList

O código de exemplo a seguir mostra como usar DeleteContactList.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteContactList](#) na Referência AWS SDK para .NET da API.

DeleteEmailIdentity

O código de exemplo a seguir mostra como usar DeleteEmailIdentity.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [DeleteEmailIdentity](#) a Referência AWS SDK para .NET da API.

DeleteEmailTemplate

O código de exemplo a seguir mostra como usar DeleteEmailTemplate.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteEmailTemplate](#) na Referência AWS SDK para .NET da API.

ListContacts

O código de exemplo a seguir mostra como usar ListContacts.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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>();
}
```

- Para obter detalhes da API, consulte [ListContacts](#) na Referência AWS SDK para .NET da API.

SendEmail

O código de exemplo a seguir mostra como usar SendEmail.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}
```

- Para obter detalhes da API, consulte [SendEmail](#) a Referência AWS SDK para .NET da API.

Cenários

Cenário de boletim informativo

O exemplo de código a seguir mostra como executar o cenário do boletim informativo da API v2 do Amazon SES.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute o cenário.

```
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);
    }
}

```

Wrapper para operações de serviço.

```

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;  
    }  
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateContact](#)
 - [CreateContactList](#)
 - [CreateEmailIdentity](#)
 - [CreateEmailTemplate](#)
 - [DeleteContactList](#)
 - [DeleteEmailIdentity](#)
 - [DeleteEmailTemplate](#)
 - [ListContacts](#)
 - [SendEmail.simples](#)
 - [SendEmail.modelo](#)

Exemplos do Amazon SNS usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon SNS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)

- [Cenários](#)
- [Exemplos sem servidor](#)

Conceitos básicos

Olá, Amazon SNS

O exemplo de código a seguir mostra como começar a usar o Amazon SNS.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
        }
    }
}
```

```
}  
}  
}
```

- Para obter detalhes da API, consulte [ListTopics](#) na Referência AWS SDK para .NET da API.

Ações

CheckIfPhoneNumberIsOptedOut

O código de exemplo a seguir mostra como usar `CheckIfPhoneNumberIsOptedOut`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}

```

- Para obter detalhes da API, consulte [CheckIfPhoneNumberIsOptedOut](#) Referência AWS SDK para .NET da API.

CreateTopic

O código de exemplo a seguir mostra como usar CreateTopic.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie um tópico com um nome específico.

```
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;
    }
}

```

Crie um tópico com um nome e atributos específicos de FIFO e desduplicação.

```

    /// <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;
}

```

- Para obter detalhes da API, consulte [CreateTopic](#) Referência AWS SDK para .NET da API.

DeleteTopic

O código de exemplo a seguir mostra como usar DeleteTopic.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Exclua um tópico por meio do respectivo ARN.

```

/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteTopica](#) Referência AWS SDK para .NET da API.

GetTopicAttributes

O código de exemplo a seguir mostra como usar GetTopicAttributes.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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");
        }
    }
}

```

- Para obter detalhes da API, consulte [GetTopicAttributes](#) a Referência AWS SDK para .NET da API.

ListSubscriptions

O código de exemplo a seguir mostra como usar ListSubscriptions.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
    }
}
}
```

- Para obter detalhes da API, consulte [ListSubscriptions](#) a Referência AWS SDK para .NET da API.

ListTopics

O código de exemplo a seguir mostra como usar ListTopics.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");  
        }  
    }  
}
```

- Para obter detalhes da API, consulte [ListTopics](#) na Referência AWS SDK para .NET da API.

Publish

O código de exemplo a seguir mostra como usar Publish.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Publique uma mensagem em um tópico.

```
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}");
    }
}

```

Publique uma mensagem em um tópico com opções de grupo, duplicação e atributo.

```

    /// <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);
}
}

```

Aplice as seleções do usuário à ação de publicação.

```

    /// <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;
}
```

- Consulte detalhes da API em [Publish](#) na Referência da API AWS SDK para .NET .

Subscribe

O código de exemplo a seguir mostra como usar Subscribe.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Inscrever um endereço de e-mail em um tópico.

```
/// <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;
}

```

Inscreva uma fila em um tópico com filtros opcionais.

```

/// <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;
    }
}
```

- Para obter detalhes da API, consulte [Subscribe](#) na Referência da API AWS SDK para .NET .

Unsubscribe

O código de exemplo a seguir mostra como usar Unsubscribe.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Cancele a assinatura de um tópico por meio de um ARN de assinatura.

```
/// <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;
}
```

- Consulte detalhes da API em [Unsubscribe](#) na Referência da API AWS SDK para .NET .

Cenários

Criação de uma aplicação do Amazon SNS

O exemplo de código a seguir mostra como criar uma aplicação que oferece funcionalidade de assinatura e publicação e tradução de mensagens.

SDK para .NET

Mostra como usar a API .NET do Amazon Simple Notification Service para criar uma aplicação Web com funcionalidade de assinatura e publicação. Além disso, essa aplicação de exemplo também traduz mensagens.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Amazon SNS
- Amazon Translate

Criar uma aplicação com tecnologia sem servidor para gerenciar fotos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

SDK para .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Para uma análise detalhada da origem desse exemplo, veja a publicação na [Comunidade da AWS](#).

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Publicar uma mensagem de texto SMS

O exemplo de código a seguir mostra como publicar mensagens SMS usando o Amazon SNS.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}

```

- Consulte detalhes da API em [Publish](#) na Referência da API AWS SDK para .NET .

Publicar mensagens em filas

O exemplo de código a seguir mostra como:

- Criar um tópico (FIFO ou não FIFO).
- Assinar várias filas no tópico com a opção de aplicar um filtro.
- Publicar mensagens no tópico.
- Pesquise as filas para ver as mensagens recebidas.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
/// <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}" +
            $"{queueUrl}" +
            "has been created.");

        if (i == 0)
        {
            Console.WriteLine(
                $"The queue URL is used to retrieve the queue ARN, " +
                "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 " +
                "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;
}
}

```

Crie uma classe que envolva operações do 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;
    }

```

```
}
```

Crie uma classe que envolva operações do 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;
}
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateQueue](#)

- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publicar](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Assinar](#)
- [Cancelar assinatura](#)

Exemplos sem servidor

Invocar uma função do Lambda em um acionador do Amazon SNS

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um tópico do SNS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do SNS com o Lambda usando .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;
        }
    }
}

```

Exemplos do Amazon SQS usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon SQS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Ações](#)
- [Cenários](#)
- [Exemplos sem servidor](#)

Conceitos básicos

Olá, Amazon SQS

O exemplo de código a seguir mostra como começar a usar o Amazon SQS.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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();
    }
}
```

- Para obter detalhes da API, consulte [ListQueues](#) na Referência AWS SDK para .NET da API.

Ações

CreateQueue

O código de exemplo a seguir mostra como usar CreateQueue.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Criar uma fila com um nome específico.

```
/// <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;
}
```

Crie uma fila do Amazon SQS e envie uma mensagem para ela.

```
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;
}
}
```

- Para obter detalhes da API, consulte [CreateQueue](#) a Referência AWS SDK para .NET da API.

DeleteMessage

O código de exemplo a seguir mostra como usar DeleteMessage.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Receba uma mensagem de uma fila do Amazon SQS e, em seguida, exclua a mensagem.

```
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;
}
}
```

- Para obter detalhes da API, consulte [DeleteMessage](#) a Referência AWS SDK para .NET da API.

DeleteMessageBatch

O código de exemplo a seguir mostra como usar DeleteMessageBatch.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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();
}
```

- Para obter detalhes da API, consulte [DeleteMessageBatch](#) na Referência AWS SDK para .NET da API.

DeleteQueue

O código de exemplo a seguir mostra como usar DeleteQueue.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Excluir uma fila usando seu 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;
}
```

- Para obter detalhes da API, consulte [DeleteQueue](#) Referência AWS SDK para .NET da API.

GetQueueAttributes

O código de exemplo a seguir mostra como usar GetQueueAttributes.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetQueueAttributes](#) a Referência AWS SDK para .NET da API.

GetQueueUrl

O código de exemplo a seguir mostra como usar `GetQueueUrl`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
        }
    }
}

```

- Para obter detalhes da API, consulte [GetQueueUrl](#) a Referência AWS SDK para .NET da API.

ReceiveMessage

O código de exemplo a seguir mostra como usar ReceiveMessage.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Receba mensagens de uma fila usando seu 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;
}
```

Receba uma mensagem de uma fila do Amazon SQS e, em seguida, exclua a mensagem.

```
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;
}
}
```

- Para obter detalhes da API, consulte [ReceiveMessage](#) a Referência AWS SDK para .NET da API.

SendMessage

O código de exemplo a seguir mostra como usar `SendMessage`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Crie uma fila do Amazon SQS e envie uma mensagem para ela.


```

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;
}
```

- Para obter detalhes da API, consulte [SendMessage](#) Referência AWS SDK para .NET da API.

SetQueueAttributes

O código de exemplo a seguir mostra como usar SetQueueAttributes.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Definir o atributo de política de uma fila para um tópico.

```
/// <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;
}

```

- Para obter detalhes da API, consulte [SetQueueAttributes](#) a Referência AWS SDK para .NET da API.

Cenários

Publicar mensagens em filas

O exemplo de código a seguir mostra como:

- Criar um tópico (FIFO ou não FIFO).

- Assinar várias filas no tópico com a opção de aplicar um filtro.
- Publicar mensagens no tópico.
- Pesquise as filas para ver as mensagens recebidas.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
/// <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;
}
}

```

Crie uma classe que envolva operações do 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;
    }
}

```

Crie uma classe que envolva operações do 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;
}
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)

- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publicar](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Assinar](#)
- [Cancelar assinatura](#)

Use a estrutura de processamento de AWS mensagens para.NET com o Amazon SQS

O exemplo de código a seguir mostra como criar aplicativos que publicam e recebem mensagens do Amazon SQS usando o AWS Message Processing Framework para.NET.

SDK para .NET

Fornece um tutorial para o AWS Message Processing Framework para.NET. O tutorial cria uma aplicação web que permite ao usuário publicar uma mensagem do Amazon SQS e uma aplicação de linha de comando que recebe a mensagem.

Para obter o código-fonte completo e instruções sobre como configurar e executar, consulte o [tutorial completo](#) no Guia do AWS SDK para .NET desenvolvedor e o exemplo em [GitHub](#).

Serviços usados neste exemplo

- Amazon SQS

Exemplos sem servidor

Invocar uma função do Lambda em um trigger do Amazon SQS

O exemplo de código a seguir mostra como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de uma fila do SQS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Consumir um evento do SQS com o Lambda usando .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;
        }
    }
}

```

Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

O exemplo de código a seguir mostra como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de uma fila do SQS. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos [Exemplos sem servidor](#).

Relatar falhas de itens em lote do SQS com o Lambda usando o .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;
    }
}

```

Exemplos de Step Functions usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Step Functions.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá, Step Functions

O exemplo de código a seguir mostra como começar a usar o Step Functions.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.");
}
}
```

- Para obter detalhes da API, consulte [ListStateMachines](#) na Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar uma atividade.
- Criar uma máquina de estado a partir de uma definição da Amazon States Language que contenha a atividade criada anteriormente como uma etapa.
- Executar a máquina de estado e responder à atividade com entrada do usuário.
- Obtenha o status e a saída finais após a conclusão da execução e, em seguida, limpe os recursos.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
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.Write("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.Write("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($"\\t{action}"));
    Console.Write($"\\n{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.Write("\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);
    }
}

```

Defina uma classe que envolva ações de máquina de estado e atividade.

```

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;
    }
}

```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [CreateActivity](#)
 - [CreateStateMachine](#)
 - [DeleteActivity](#)
 - [DeleteStateMachine](#)
 - [DescribeExecution](#)
 - [DescribeStateMachine](#)
 - [GetActivityTask](#)
 - [ListActivities](#)
 - [ListStateMachines](#)
 - [SendTaskSuccess](#)
 - [StartExecution](#)

- [StopExecution](#)

Ações

CreateActivity

O código de exemplo a seguir mostra como usar CreateActivity.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateActivity](#) a Referência AWS SDK para .NET da API.

CreateStateMachine

O código de exemplo a seguir mostra como usar CreateStateMachine.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [CreateStateMachine](#) a Referência AWS SDK para .NET da API.

DeleteActivity

O código de exemplo a seguir mostra como usar DeleteActivity.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DeleteActivity](#) a Referência AWS SDK para .NET da API.

DeleteStateMachine

O código de exemplo a seguir mostra como usar DeleteStateMachine.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [DeleteStateMachine](#) na Referência AWS SDK para .NET da API.

DescribeExecution

O código de exemplo a seguir mostra como usar DescribeExecution.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [DescribeExecution](#) na Referência AWS SDK para .NET da API.

DescribeStateMachine

O código de exemplo a seguir mostra como usar `DescribeStateMachine`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeStateMachine](#) na Referência AWS SDK para .NET da API.

GetActivityTask

O código de exemplo a seguir mostra como usar `GetActivityTask`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetActivityTask](#) Referência AWS SDK para .NET da API.

ListActivities

O código de exemplo a seguir mostra como usar `ListActivities`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListActivities](#) na Referência AWS SDK para .NET da API.

ListExecutions

O código de exemplo a seguir mostra como usar `ListExecutions`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListExecutions](#) na Referência AWS SDK para .NET da API.

ListStateMachines

O código de exemplo a seguir mostra como usar ListStateMachines.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListStateMachines](#) na Referência AWS SDK para .NET da API.

SendTaskSuccess

O código de exemplo a seguir mostra como usar SendTaskSuccess.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [SendTaskSuccess](#) a Referência AWS SDK para .NET da API.

StartExecution

O código de exemplo a seguir mostra como usar `StartExecution`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [StartExecution](#) na Referência AWS SDK para .NET da API.

AWS STS exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with AWS STS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

AssumeRole

O código de exemplo a seguir mostra como usar AssumeRole.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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.USSouth2;

    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}");
    }
}
}
```

- Para obter detalhes da API, consulte [AssumeRole](#) a Referência AWS SDK para .NET da API.

Suporte exemplos usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET with Suporte.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Conceitos básicos](#)
- [Conceitos básicos](#)
- [Ações](#)

Conceitos básicos

Olá Suporte

O exemplo de código a seguir mostra como começar a usar o Suporte.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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($"\\tHello AWS Support! There are {response.Services.Count}
        services available.");
    }
}
```

- Para obter detalhes da API, consulte [DescribeServices](#) a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Obter e exibir os serviços disponíveis e os níveis de gravidade dos casos.
- Criar um caso de suporte usando um serviço, uma categoria e um nível de gravidade selecionados.
- Obter e exibir uma lista de casos em aberto para o dia atual.
- Adicionar um conjunto de anexos e uma comunicação ao novo caso.
- Descrever o novo anexo e a comunicação para o caso.
- Resolver o caso.
- Obtenha e exiba uma lista de casos resolvidos para o dia atual.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

Execute um cenário interativo em um prompt de comando.

```
/// <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($"1. Displaying first 10 services:");
        for (int i = 0; i < 10 && i < services.Count; i++)
        {
            Console.WriteLine($"\\t{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($"{t{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));
    }
}

```

Métodos de embalagem usados pelo cenário para Suporte ações.

```

/// <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 { }
}

```

```
    }  
    catch (Amazon.AWSSupport.AmazonAWSSupportException ex)  
    {  
        if (ex.ErrorCode == "SubscriptionRequiredException")  
        {  
            return false;  
        }  
        else throw;  
    }  
}  
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)
 - [DescribeCases](#)
 - [DescribeCommunications](#)
 - [DescribeServices](#)
 - [DescribeSeverityLevels](#)
 - [ResolveCase](#)

Ações

AddAttachmentsToSet

O código de exemplo a seguir mostra como usar AddAttachmentsToSet.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }

```

- Para obter detalhes da API, consulte [AddAttachmentsToSet](#) Referência AWS SDK para .NET da API.

AddCommunicationToCase

O código de exemplo a seguir mostra como usar AddCommunicationToCase.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [AddCommunicationToCase](#) a Referência AWS SDK para .NET da API.

CreateCase

O código de exemplo a seguir mostra como usar CreateCase.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
    }
```

- Para obter detalhes da API, consulte [CreateCase](#) a Referência AWS SDK para .NET da API.

DescribeAttachment

O código de exemplo a seguir mostra como usar DescribeAttachment.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeAttachment](#) a Referência AWS SDK para .NET da API.

DescribeCases

O código de exemplo a seguir mostra como usar DescribeCases.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [DescribeCases](#) a Referência AWS SDK para .NET da API.

DescribeCommunications

O código de exemplo a seguir mostra como usar DescribeCommunications.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [DescribeCommunications](#) na Referência AWS SDK para .NET da API.

DescribeServices

O código de exemplo a seguir mostra como usar `DescribeServices`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}
```

- Para obter detalhes da API, consulte [DescribeServices](#) a Referência AWS SDK para .NET da API.

DescribeSeverityLevels

O código de exemplo a seguir mostra como usar `DescribeSeverityLevels`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [DescribeSeverityLevels](#) na Referência AWS SDK para .NET da API.

ResolveCase

O código de exemplo a seguir mostra como usar `ResolveCase`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;  
}
```

- Para obter detalhes da API, consulte [ResolveCase](#) Referência AWS SDK para .NET da API.

Exemplos do Amazon Textract usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Textract.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Cenários](#)

Cenários

Criar uma aplicação para analisar o feedback dos clientes

O exemplo de código a seguir mostra como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

SDK para .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Exemplos do Amazon Transcribe usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Transcribe.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)

Ações

CreateVocabulary

O código de exemplo a seguir mostra como usar CreateVocabulary.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
}

```

- Para obter detalhes da API, consulte [CreateVocabulary](#) a Referência AWS SDK para .NET da API.

DeleteMedicalTranscriptionJob

O código de exemplo a seguir mostra como usar DeleteMedicalTranscriptionJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

/// <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;
    }

```

- Para obter detalhes da API, consulte [DeleteMedicalTranscriptionJob](#) a Referência AWS SDK para .NET da API.

DeleteTranscriptionJob

O código de exemplo a seguir mostra como usar DeleteTranscriptionJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
}
```

- Para obter detalhes da API, consulte [DeleteTranscriptionJob](#) a Referência AWS SDK para .NET da API.

DeleteVocabulary

O código de exemplo a seguir mostra como usar DeleteVocabulary.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

```
}
```

- Para obter detalhes da API, consulte [DeleteVocabulary](#) a Referência AWS SDK para .NET da API.

GetTranscriptionJob

O código de exemplo a seguir mostra como usar `GetTranscriptionJob`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetTranscriptionJob](#) a Referência AWS SDK para .NET da API.

GetVocabulary

O código de exemplo a seguir mostra como usar `GetVocabulary`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [GetVocabulary](#) a Referência AWS SDK para .NET da API.

ListMedicalTranscriptionJobs

O código de exemplo a seguir mostra como usar `ListMedicalTranscriptionJobs`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListMedicalTranscriptionJobs](#) na Referência AWS SDK para .NET da API.

ListTranscriptionJobs

O código de exemplo a seguir mostra como usar ListTranscriptionJobs.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListTranscriptionJobs](#) na Referência AWS SDK para .NET da API.

ListVocabularies

O código de exemplo a seguir mostra como usar `ListVocabularies`.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [ListVocabularies](#) na Referência AWS SDK para .NET da API.

StartMedicalTranscriptionJob

O código de exemplo a seguir mostra como usar StartMedicalTranscriptionJob.

SDK para .NET

 Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}

```

- Para obter detalhes da API, consulte [StartMedicalTranscriptionJob](#) a Referência AWS SDK para .NET da API.

StartTranscriptionJob

O código de exemplo a seguir mostra como usar StartTranscriptionJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```

    /// <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;
    }
```

- Para obter detalhes da API, consulte [StartTranscriptionJob](#) na Referência AWS SDK para .NET da API.

UpdateVocabulary

O código de exemplo a seguir mostra como usar UpdateVocabulary.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
/// <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;
}
```

- Para obter detalhes da API, consulte [UpdateVocabulary](#) a Referência AWS SDK para .NET da API.

Exemplos do Amazon Translate usando SDK para .NET

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET com o Amazon Translate.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- [Ações](#)
- [Cenários](#)

Ações

DescribeTextTranslationJob

O código de exemplo a seguir mostra como usar `DescribeTextTranslationJob`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    }
}
```

- Para obter detalhes da API, consulte [DescribeTextTranslationJob](#) Referência AWS SDK para .NET da API.

ListTextTranslationJobs

O código de exemplo a seguir mostra como usar `ListTextTranslationJobs`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
    });
}
```

- Para obter detalhes da API, consulte [ListTextTranslationJobs](#) na Referência AWS SDK para .NET da API.

StartTextTranslationJob

O código de exemplo a seguir mostra como usar `StartTextTranslationJob`.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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;
}
}

```

- Para obter detalhes da API, consulte [StartTextTranslationJob](#) na Referência AWS SDK para .NET da API.

StopTextTranslationJob

O código de exemplo a seguir mostra como usar StopTextTranslationJob.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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}");
        }
    }
}

```

- Para obter detalhes da API, consulte [StopTextTranslationJob](#) Referência AWS SDK para .NET da API.

TranslateText

O código de exemplo a seguir mostra como usar TranslateText.

SDK para .NET

Note

Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no [AWS Code Examples Repository](#).

```
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);
    }
}

```

- Para obter detalhes da API, consulte [TranslateText](#) Referência AWS SDK para .NET da API.

Cenários

Criação de uma aplicação do Amazon SNS

O exemplo de código a seguir mostra como criar uma aplicação que oferece funcionalidade de assinatura e publicação e tradução de mensagens.

SDK para .NET

Mostra como usar a API .NET do Amazon Simple Notification Service para criar uma aplicação Web com funcionalidade de assinatura e publicação. Além disso, essa aplicação de exemplo também traduz mensagens.

Para obter o código-fonte completo e instruções sobre como configurar e executar, veja o exemplo completo em [GitHub](#).

Serviços usados neste exemplo

- Amazon SNS
- Amazon Translate

Criar uma aplicação para analisar o feedback dos clientes

O exemplo de código a seguir mostra como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

SDK para .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em [GitHub](#).

Serviços usados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Segurança para este AWS produto ou serviço

A segurança da nuvem na Amazon Web Services (AWS) é a nossa maior prioridade. Como cliente da AWS, você contará com um data center e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança. A segurança é uma responsabilidade compartilhada entre você AWS e você. O [modelo de responsabilidade compartilhada](#) descreve isso como a Segurança da nuvem e a Segurança na nuvem.

Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa todos os serviços oferecidos na AWS nuvem e fornecer serviços que você possa usar com segurança. Nossa responsabilidade de segurança é a maior prioridade em AWS, e a eficácia de nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos [Programas de AWS Conformidade](#).

Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você está usando e por outros fatores, incluindo a sensibilidade de seus dados, os requisitos da sua organização e as leis e regulamentos aplicáveis.

Esse AWS produto ou serviço segue o [modelo de responsabilidade compartilhada](#) por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do [AWS serviço](#), consulte a [página de documentação de segurança](#) do serviço e os [AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade](#).

Tópicos

- [Proteção de dados neste AWS produto ou serviço](#)
- [Gerenciamento de Identidade e Acesso](#)
- [Validação de conformidade para este AWS produto ou serviço](#)
- [Resiliência para este AWS produto ou serviço](#)
- [Segurança da infraestrutura para este AWS produto ou serviço](#)
- [Aplicando uma versão mínima do TLS no AWS SDK para .NET](#)
- [Migração do cliente de criptografia Amazon S3 \(V1 para V2\)](#)
- [Migração do cliente de criptografia Amazon S3 \(V2 para V4\)](#)

Proteção de dados neste AWS produto ou serviço

O [modelo de responsabilidade AWS compartilhada](#) de se aplica à proteção de dados neste AWS produto ou serviço. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para saber mais sobre a privacidade de dados, consulte as [Data Privacy FAQ](#). Para saber mais sobre a proteção de dados na Europa, consulte a postagem do blog [AWS Shared Responsibility Model and RGPD](#) no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como [trabalhar com CloudTrail trilhas](#) no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para saber mais sobre os endpoints FIPS disponíveis, consulte [Federal Information Processing Standard \(FIPS\) 140-3](#).

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com este AWS produto, serviço ou outro Serviços da AWS usando o console, a API ou AWS SDKs. AWS CLI Quaisquer dados inseridos em tags ou em campos de texto

de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Gerenciamento de Identidade e Acesso

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar AWS os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

- [Público](#)
- [Autenticação com identidades](#)
- [Gerenciar o acesso usando políticas](#)
- [Como Serviços da AWS trabalhar com o IAM](#)
- [Solução de problemas AWS de identidade e acesso](#)

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz AWS.

Usuário do serviço — Se você Serviços da AWS costuma fazer seu trabalho, seu administrador fornece as credenciais e as permissões de que você precisa. À medida que você usa mais AWS recursos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se você não conseguir acessar um recurso no AWS, consulte [Solução de problemas AWS de identidade e acesso](#) o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador de serviços — Se você é responsável pelos AWS recursos da sua empresa, provavelmente tem acesso total AWS a. É seu trabalho determinar quais AWS recursos e recursos seus usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com AWS, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador do IAM: se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar o acesso ao AWS. Para ver exemplos de políticas AWS baseadas em identidade que você pode usar no IAM, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Autenticação com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte [Como fazer login em sua Conta da AWS](#) no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte [AWS Signature Version 4 para solicitações de API](#) no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em [Tarefas que exigem credenciais de usuário-raiz](#) no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários humanos usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório corporativo, provedor de identidade da web ou Directory Service que acessa Serviços da AWS usando credenciais de uma fonte de identidade. As identidades federadas assumem funções que oferecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos Centro de Identidade do AWS IAM. Para saber mais, consulte [O que é o IAM Identity Center?](#) no Guia do usuário do Centro de Identidade do AWS IAM .

Usuários e grupos do IAM

Um [usuário do IAM](#) é uma identidade com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para obter mais informações, consulte [Exigir que usuários humanos usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias](#) no Guia do usuário do IAM.

Um [grupo do IAM](#) especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte [Casos de uso de usuários do IAM](#) no Guia do usuário do IAM.

Perfis do IAM

Uma [perfil do IAM](#) é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função [mudando de um usuário para uma função do IAM \(console\)](#) ou chamando uma operação de AWS API AWS CLI ou. Para saber mais, consulte [Métodos para assumir um perfil](#) no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em [Acesso a recursos entre contas no IAM](#) no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte [Visão geral das políticas JSON](#) no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você anexa a uma identidade (usuário, grupo ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte [Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente](#) no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte [Escolher entre políticas gerenciadas e políticas em linha](#) no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. Entre os exemplos estão políticas de confiança de perfil do IAM e políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. É necessário [especificar uma entidade principal](#) em uma política baseada em recursos.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o AWS WAF Amazon VPC são exemplos de serviços que oferecem suporte. ACLs Para saber mais ACLs, consulte a [visão geral da lista de controle de acesso \(ACL\)](#) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte [Limites de permissões para identidades do IAM](#) no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — Especifique as permissões máximas para uma organização ou unidade organizacional em AWS Organizations. Para saber mais, consulte [Políticas de controle de serviço](#) no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs) — Defina o máximo de permissões disponíveis para recursos em suas contas. Para obter mais informações, consulte [Políticas de controle de recursos \(RCPs\)](#) no Guia AWS Organizations do usuário.
- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, consulte [Políticas de sessão](#) no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte [Lógica de avaliação de políticas](#) no Guia do usuário do IAM.

Como Serviços da AWS trabalhar com o IAM

Para ter uma visão de alto nível de como Serviços da AWS funciona com a maioria dos recursos do IAM, consulte [AWS os serviços que funcionam com o IAM](#) no Guia do usuário do IAM.

Para saber como usar um específico AWS service (Serviço da AWS) com o IAM, consulte a seção de segurança do Guia do usuário do serviço relevante.

Solução de problemas AWS de identidade e acesso

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com AWS um IAM.

Tópicos

- [Não estou autorizado a realizar uma ação em AWS](#)
- [Não estou autorizado a realizar iam: PassRole](#)
- [Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos](#)

Não estou autorizado a realizar uma ação em AWS

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo `my-example-widget` fictício, mas não tem as permissões `aws:GetWidget` fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso `my-example-widget` usando a ação `aws:GetWidget`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no AWS. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos

É possível criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se é AWS compatível com esses recursos, consulte [Como Serviços da AWS trabalhar com o IAM](#).
- Para saber como fornecer acesso aos seus recursos em todas as Contas da AWS que você possui, consulte Como [fornecer acesso a um usuário do IAM em outra Conta da AWS que você possui](#) no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como [fornecer acesso Contas da AWS a terceiros](#) no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte [Conceder acesso a usuários autenticados externamente \(federação de identidades\)](#) no Guia do usuário do IAM.
- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte [Acesso a recursos entre contas no IAM](#) no Guia do usuário do IAM.

Validação de conformidade para este AWS produto ou serviço

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte [Serviços da AWS Escopo por Programa de Conformidade](#) [Serviços da AWS](#) e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de [AWS conformidade Programas AWS](#) de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte [Baixar relatórios em AWS Artifact](#) .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. Para obter mais informações sobre sua responsabilidade de conformidade ao usar Serviços da AWS, consulte a [documentação AWS de segurança](#).

Esse AWS produto ou serviço segue o [modelo de responsabilidade compartilhada](#) por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do [AWS serviço, consulte a página de documentação de segurança](#) do serviço e os [AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade](#).

Resiliência para este AWS produto ou serviço

A infraestrutura AWS global é construída em torno Regiões da AWS de zonas de disponibilidade.

Regiões da AWS fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância.

Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte [Infraestrutura AWS global](#).

Esse AWS produto ou serviço segue o [modelo de responsabilidade compartilhada](#) por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do [AWS serviço, consulte a página de documentação de segurança](#) do serviço e os [AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade](#).

Segurança da infraestrutura para este AWS produto ou serviço

Esse AWS produto ou serviço usa serviços gerenciados e, portanto, é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte [AWS Cloud Security](#). Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte [Proteção](#) de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar este AWS Produto ou Serviço pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.

- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o [AWS Security Token Service](#) (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Esse AWS produto ou serviço segue o [modelo de responsabilidade compartilhada](#) por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do [AWS serviço](#), consulte a [página de documentação de segurança](#) do serviço e os [AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade](#).

Aplicando uma versão mínima do TLS no AWS SDK para .NET

Para aumentar a segurança ao se comunicar com AWS serviços, você deve configurar o AWS SDK para .NET para usar o TLS 1.2 ou posterior.

O AWS SDK para .NET usa o tempo de execução do .NET subjacente para determinar qual protocolo de segurança usar. Por padrão, as versões atuais do .NET usam o protocolo configurado mais recentemente que for compatível com o sistema operacional. Seu aplicativo pode substituir esse comportamento do SDK, mas não é recomendado fazê-lo.

.NET Core

Por padrão, o .NET Core usa o protocolo configurado mais recentemente que for compatível com o sistema operacional. O AWS SDK para .NET não fornece um mecanismo para substituir isso.

Se você estiver usando uma versão do .NET Core anterior à 2.1, recomendamos veementemente que atualize sua versão do .NET Core.

Consulte os itens a seguir para obter informações específicas sobre cada sistema operacional.

Windows

As distribuições modernas do Windows têm o suporte para TLS 1.2 [habilitado por padrão](#). Se você estiver executando o Windows 7 SP1 ou o Windows Server 2008 R2 SP1, precisará garantir que o suporte ao TLS 1.2 esteja habilitado no registro, conforme descrito em <https://learn.microsoft.com/en->

[us/windows-server/security/tls/tls-registry-settings#tls-12](#). Se estiver executando uma distribuição anterior, você deverá atualizar seu sistema operacional. Para obter informações sobre o suporte ao TLS 1.3 no Windows, consulte a documentação mais recente da Microsoft para ver as versões mínimas exigidas de cliente ou servidor.

macOS

Se estiver executando o .NET Core 2.1 ou posterior, o TLS 1.2 estará habilitado por padrão. O TLS 1.2 é compatível com o [OS X Mavericks v10.9 ou posterior](#). [.NET Core versão 2.1 e versões posteriores exigem versões mais recentes do macOS, conforme descrito em? https://learn.microsoft.com/en-us/dotnet/core/install/windows-tabs=net80&pivots=os-macos](#).

Caso você esteja usando o .NET Core 1.0, o .NET Core [usa OpenSSL no macOS](#), uma dependência que deve ser instalada separadamente. O OpenSSL adicionou suporte para TLS 1.2 na versão 1.0.1 e adicionou suporte para TLS 1.3 na versão 1.1.1.

Linux

O .NET Core no Linux requer OpenSSL, que é fornecido junto com muitas distribuições Linux. Mas também pode ser instalado separadamente. O OpenSSL adicionou suporte para TLS 1.2 na versão 1.0.1 e adicionou suporte para TLS 1.3 na versão 1.1.1. Caso você esteja usando uma versão moderna do .NET Core (2.1 ou posterior) e tenha instalado um gerenciador de pacotes, é provável que uma versão mais moderna do OpenSSL tenha sido instalada para você.

Para ter certeza, você pode executar o **openssl version** em um terminal e verificar se a versão é posterior à 1.0.1.

NET Framework

Se você estiver executando uma versão moderna do .NET Framework (4.7 ou posterior) e uma versão moderna do Windows (pelo menos o Windows 8 para clientes, o Windows Server 2012 ou posterior para servidores), o TLS 1.2 será habilitado e usado por padrão.

Se você estiver usando um runtime do .NET Framework que não usa as configurações do sistema operacional (.NET Framework 3.5 a 4.5.2), eles AWS SDK para .NET tentarão [adicionar suporte para TLS 1.1 e TLS 1.2](#) aos protocolos suportados. Se você estiver usando o .NET Framework 3.5, esse processo só será bem-sucedido se o hot patch apropriado estiver instalado, da seguinte forma:

- Windows 10 versão 1511 e Windows Server 2016 — [KB3156421](#)
- Windows 8.1 e Windows Server 2012 R2 — [KB3154520](#)

- Windows Server 2012 — [KB3154519](#)
- Windows 7 SP1 e Server 2008 R2 SP1 — [KB3154518](#)

Warning

A partir de 15 de agosto de 2024, o suporte para o .NET Framework 3.5 AWS SDK para .NET será encerrado e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

Se seu aplicativo estiver sendo executado em um novo .NET Framework no Windows 7 SP1 ou no Windows Server 2008 R2 SP1, você precisará garantir que o suporte ao TLS 1.2 esteja habilitado no registro, conforme descrito em <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12>. Nas versões mais recentes do Windows, ele está [habilitado por padrão](#).

Para obter as melhores práticas detalhadas para usar o TLS com o .NET Framework, consulte o artigo da Microsoft em <https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls>.

Ferramentas da AWS para PowerShell

[Ferramentas da AWS para PowerShell](#) use o AWS SDK para .NET para todas as chamadas para AWS serviços. O comportamento do seu ambiente depende da versão do Windows PowerShell que você está executando, da seguinte forma.

Windows PowerShell 2.0 a 5.x

O Windows PowerShell 2.0 a 5.x é executado no .NET Framework. Você pode verificar qual runtime do .NET (2.0 ou 4.0) está sendo usado PowerShell usando o comando a seguir.

```
$PSVersionTable.CLRVersion
```

- Ao usar o .NET Runtime 2.0, siga as instruções fornecidas anteriormente em relação ao AWS SDK para .NET e ao .NET Framework 3.5.

Warning

A partir de 15 de agosto de 2024, o suporte para o .NET Framework 3.5 AWS SDK para .NET será encerrado e a versão mínima do .NET Framework será alterada para 4.7.2.

Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

- Ao usar o .NET Runtime 4.0, siga as instruções fornecidas anteriormente em relação ao AWS SDK para .NET e ao .NET Framework 4+.

Windows PowerShell 6.0

O Windows PowerShell 6.0 e versões mais recentes são executados no .NET Core. Você pode verificar qual versão do .NET Core está sendo usada executando o comando a seguir.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.FrameworkName], $true).FrameworkName
```

Siga as instruções fornecidas anteriormente em relação à versão relevante do .NET Core AWS SDK para .NET e à respectiva.

Xamarin

Para o Xamarin, veja as instruções em [-layer-security. https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport](https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport) Em resumo:

Para Android

- Requer o Android 5.0 ou posterior.
- Propriedades do projeto, opções do Android: a HttpClient implementação deve ser definida como Android e a SSL/TLS implementação definida como TLS 1.2+ nativo.

Para iOS

- Requer o iOS 7 ou posterior.
- Propriedades do projeto, versão iOS: HttpClient a implementação deve ser definida como NSUrlSessão.

No macOS

- Requer o macOS 10.9 ou posterior.

- Opções do projeto, compilação, compilação do Mac: HttpClient a implementação deve ser definida como NSURLSessão.

Unity

Você deve usar o Unity 2018.2 ou posterior e usar o tempo de execução de scripts equivalente ao .NET 4.x. Você pode definir isso em Configurações do projeto, Configuração, Player, conforme descrito em <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html>. O tempo de execução de script equivalente do .NET 4.x permite o suporte ao TLS 1.2 em todas as plataformas Unity que executam Mono ou CPP. IL2

Navegador (para Blazor) WebAssembly

WebAssembly é executado no navegador em vez de no servidor e usa o navegador para lidar com o tráfego HTTP. Portanto, o suporte a TLS é determinado pelo suporte do navegador.

[O Blazor WebAssembly, na versão prévia do ASP.NET Core 3.1, é suportado somente em navegadores que oferecem suporte WebAssembly, conforme descrito em -platforms. https://learn.microsoft.com/en-us/aspnet/core/blazor/supported](https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms) Todos os principais navegadores eram compatíveis com o TLS 1.2 antes do suporte. WebAssembly Se esse for o caso do seu navegador, se o seu aplicativo for executado, ele poderá se comunicar por TLS 1.2.

Consulte a documentação do seu navegador para obter mais informações e verificação.

Migração do cliente de criptografia Amazon S3 (V1 para V2)

Note

Se você estiver usando a V2 e quiser migrar para a V4, consulte. [Migração do cliente de criptografia Amazon S3 \(V2 para V4\)](#)

Este tópico mostra como migrar suas aplicações da Versão 1 (V1) do cliente de criptografia do Amazon Simple Storage Service (Amazon S3) para a Versão 2 (V2) e garantir a disponibilidade da aplicação durante todo o processo de migração.

Objetos criptografados com o cliente V2 não podem ser descriptografados com o cliente V1. Para facilitar a migração para o novo cliente sem precisar recriptografar todos os objetos de uma vez,

foi fornecido um cliente “V1 de transição”. Esse cliente pode descriptografar objetos criptografados com a V1 e V2, mas criptografa somente objetos em formato compatível com a V1. O cliente V2 pode descriptografar objetos criptografados em V1 e V2 (quando habilitado para objetos V1), mas criptografa objetos somente em formato compatível com V2.

Visão geral da migração

Essa migração acontece em três fases: Essas fases são apresentadas aqui e serão descritas em detalhes posteriormente. Cada fase deve ser concluída para todos os clientes que usam objetos compartilhados antes que a próxima fase seja iniciada.

1. Atualize os clientes existentes para clientes V1 de transição para ler novos formatos. Primeiro, atualize seus aplicativos para que dependam do cliente V1 de transição em vez do cliente V1. O cliente V1 de transição permite que seu código existente descriptografe objetos escritos pelos novos clientes V2 e objetos escritos em formato compatível com a V1.

Note

O cliente V1 de transição é fornecido somente para fins de migração. prossiga com a atualização para o cliente V2 depois de passar para o cliente de transição V1.

2. Migre clientes V1 de transição para clientes V2 para escrever novos formatos. Em seguida, substitua todos os clientes V1 de transição em seus aplicativos por clientes V2 e defina o perfil de segurança como V2AndLegacy. Definir esse perfil de segurança em clientes V2 permite que esses clientes descriptografem objetos que foram criptografados em formato compatível com V1.
3. Atualize os clientes V2 para que não leiam mais os formatos V1. Por fim, depois que todos os clientes tiverem migrado para a V2 e todos os objetos tiverem sido criptografados ou recriptografados em um formato compatível com a V2, defina o perfil de segurança da V2 como V2 em vez de V2AndLegacy. Isso evita a descriptografia de objetos que estão no formato compatível com V1.

Atualize os clientes existentes para clientes V1 de transição para ler novos formatos.

O cliente de criptografia V2 usa algoritmos de criptografia incompatíveis com as versões mais antigas do cliente. A primeira etapa da migração é atualizar seus clientes de descriptografia V1 para que eles possam ler o novo formato.

O cliente V1 de transição permite que seus aplicativos descriptografem objetos criptografados na V1 e V2. Esse cliente faz parte do pacote [NuGet Amazon.Extensions.S3.Encryption](#). Execute as etapas a seguir em cada um dos seus aplicativos para usar o cliente V1 de transição.

1. Adquira uma nova dependência do pacote [Amazon.Extensions.S3.Encryption](#). Se o seu projeto depende diretamente do AWSSDK.S3 ou. AWSSDK KeyManagementService pacotes, você deve atualizar essas dependências ou removê-las para que suas versões atualizadas sejam inseridas com esse novo pacote.
2. Altere a declaração `using` apropriada de `Amazon.S3.Encryption` para `Amazon.Extensions.S3.Encryption`, da seguinte forma:

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. Reconstrua e reimplante o aplicativo.

O cliente V1 de transição é totalmente compatível com a API do cliente V1, portanto, nenhuma outra alteração de código é necessária.

Migre clientes V1 de transição para clientes V2 para escrever novos formatos.

O cliente V2 faz parte do pacote [NuGet Amazon.Extensions.S3.Encryption](#). Ele permite que seus aplicativos descriptografem objetos criptografados na V1 e V2 (se configurados para fazer isso), mas criptografa objetos somente em formato compatível com V2.

Depois de atualizar seus clientes existentes para ler o novo formato de criptografia, você poderá prosseguir com a atualização segura de seus aplicativos para os clientes de criptografia e descriptografia V2. Execute as etapas a seguir em cada um dos seus aplicativos para usar o cliente V2.

1. Fazer `EncryptionMaterials` alteração `EncryptionMaterialsV2`.
 - a. Ao usar o KMS:
 - i. Forneça um ID da chave do KMS.
 - ii. Declare o método de criptografia que você está usando; ou seja, `KmsType.KmsContext`.

- iii. Forneça um contexto de criptografia ao KMS para associar a essa chave de dados. Você pode enviar um dicionário vazio (o contexto de criptografia da Amazon ainda será mesclado), mas é recomendável fornecer contexto adicional.
 - b. Ao usar métodos de wrapping da chave fornecidos pelo usuário (criptografia simétrica ou assimétrica):
 - i. Forneça uma instância AES ou RSA que contenha os materiais de criptografia.
 - ii. Declare qual algoritmo de criptografia usar; ou seja, `SymmetricAlgorithmType.AesGcm` ou `AsymmetricAlgorithmType.RsaOaepSha1`.
2. Altere `AmazonS3CryptoConfiguration` para `AmazonS3CryptoConfigurationV2` com a propriedade `SecurityProfile` definida como `SecurityProfile.V2AndLegacy`.
3. Fazer `AmazonS3EncryptionClient` alteração `AmazonS3EncryptionClientV2`. Esse cliente pega os objetos `AmazonS3CryptoConfigurationV2` e `EncryptionMaterialsV2` recém-convertidos das etapas anteriores.

Exemplo: KMS para KMS+Context

Pré-migração

```
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);
```

Pós-migração

```
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);
```

Exemplo: Algoritmo simétrico (AES-CBC para AES-GCM Key Wrap)

StorageMode pode ser ObjectMetadata ou InstructionFile.

Pré-migração

```
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);
```

Pós-migração

```
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

Ao descriptografar com o AES-GCM, leia o objeto inteiro até o fim antes de começar a usar os dados descriptografados. Isso é para verificar se o objeto não foi modificado desde que foi criptografado.

Exemplo: Algoritmo assimétrico (RSA para RSA-OAEP-SHA 1 quebra de chave)

StorageMode pode ser ObjectMetadata ou InstructionFile.

Pré-migração

```
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);
```

Pós-migração

```
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);
```

Atualize os clientes V2 para que não leiam mais os formatos V1.

Eventualmente, todos os objetos terão sido criptografados ou recriptografados usando um cliente V2. Depois que essa conversão for concluída, você poderá desativar a compatibilidade com a V1 nos clientes V2 definindo a propriedade `SecurityProfile` como `SecurityProfile.V2`, conforme mostrado no trecho a seguir.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);  
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migração do cliente de criptografia Amazon S3 (V2 para V4)

Note

Se você estiver usando a V1 e quiser migrar para a V4, primeiro deverá migrar para a V2. Consulte [Migração do cliente de criptografia Amazon S3 \(V1 para V2\)](#).

Este tópico mostra como migrar seus aplicativos da versão 2 (V2) do cliente de criptografia Amazon Simple Storage Service (Amazon S3) para a versão 4 (V4) e garantir a disponibilidade dos aplicativos durante todo o processo de migração. A V4 usa o AES-GCM com o compromisso principal de criptografia de conteúdo e apresenta políticas de compromisso para aumentar a segurança contra ataques de substituição de chaves.

O cliente V4 está disponível no pacote [NuGet Amazon.Extensions.S3.Encryption](#).

Important

Alterações importantes: clientes V4 configurados com `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` política não podem descriptografar objetos criptografados com clientes V1 ou V2. Somente os clientes V2 mais recentes podem descriptografar objetos criptografados em V4 com comprometimento de chave. Antes de passar para a `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` política, você deve criptografar novamente todos os seus dados existentes usando clientes V4 com o compromisso de chave ativado.

Compreendendo os conceitos da V4

A V4 usa o AES-GCM com o compromisso fundamental de criptografia de conteúdo e apresenta os principais conceitos de segurança que aprimoram a proteção de seus dados criptografados:

Política de compromisso

A Política de Compromisso controla como o cliente de criptografia lida com o comprometimento da chave durante as operações de criptografia e descriptografia. A V4 oferece suporte a três políticas de compromisso:

FORBID_ENCRYPT_ALLOW_DECRYPT

Criptografia: sem compromisso

Descriptografia: permite que objetos não confirmem

Segurança: não impõe o compromisso e pode permitir a adulteração

Compatibilidade: todas as implementações V2 e V4 podem ler objetos criptografados com esta política

REQUIRE_ENCRYPT_ALLOW_DECRYPT

Criptografia: com compromisso fundamental

Descriptografia: permite que objetos confirmados e não confirmados

Segurança: novos objetos são protegidos contra ataques de substituição de chaves, objetos antigos ainda podem ser lidos

Compatibilidade: somente a V4 oferece suporte a essa política

REQUIRE_ENCRYPT_REQUIRE_DECRYPT (Padrão para V4)

Criptografia: com compromisso fundamental

Descriptografia: somente confirmando objetos

Segurança: cumprimento total do compromisso para máxima segurança

Compatibilidade: somente a V4 oferece suporte a essa política

AES GCM com compromisso fundamental

A V4 usa o AES-GCM com o compromisso fundamental de criptografia de conteúdo, o que fornece segurança aprimorada:

- Proteção contra adulteração: protege contra ataques de substituição de chave vinculando criptograficamente a chave aos dados criptografados.
- Compatibilidade de versão: objetos criptografados com comprometimento de chave só podem ser descriptografados por clientes V4 e versões mais recentes.

Warning

Antes de ativar a criptografia de comprometimento de chave na produção, certifique-se de que todos os aplicativos que precisam descriptografar seus objetos tenham sido atualizados para a V4 ou mais recente, pois o cliente V2 está se tornando obsoleto.

Atualize os clientes existentes para ler os formatos V4

O cliente de criptografia V4 usa algoritmos de criptografia que as versões mais antigas do cliente não suportam. A primeira etapa da migração é atualizar seus clientes V2 para que eles possam ler o novo formato V4.

Atualizar dependências NuGet do Package

Atualize seus aplicativos para usar a versão mais recente do pacote [NuGet Amazon.Extensions.S3.Encryption](#) que inclui suporte à V4. Execute as seguintes etapas em cada um dos seus aplicativos:

1. Atualize para o pacote [Amazon.Extensions.S3.Encryption](#) mais recente. Se o seu projeto depende diretamente do AWSSDK.S3 ou AWSSDK KeyManagementService pacotes, você deve atualizar essas dependências ou removê-las para que suas versões atualizadas sejam inseridas com esse novo pacote.
2. Certifique-se de que suas `using` declarações façam referência ao namespace correto:

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;
```

3. Reconstrua e reimplante o aplicativo.

Seus clientes V2 existentes continuarão trabalhando com o pacote atualizado e poderão descriptografar objetos criptografados por clientes V4 (dependendo da política de compromisso usada).

Crie e implante aplicativos

Depois de atualizar as dependências NuGet do pacote:

1. Crie seu aplicativo para garantir que todas as dependências sejam resolvidas corretamente.
2. Teste seu aplicativo em um ambiente de desenvolvimento para verificar se a funcionalidade existente continua funcionando.
3. Implante o aplicativo atualizado em seu ambiente de produção.

Essa atualização permite que seus clientes V2 existentes descriptografem objetos que serão criptografados por clientes V4, garantindo compatibilidade durante o processo de migração.

Migre para clientes V4

Depois de atualizar seus clientes existentes para ler o novo formato de criptografia, você pode continuar atualizando seus aplicativos com segurança para usar clientes de criptografia e descriptografia V4. O cliente V4 fornece segurança aprimorada por meio do comprometimento de chaves, mantendo a compatibilidade com objetos criptografados existentes.

Processo de migração em 4 etapas

A migração da V2 para a V4 segue um processo estruturado de 4 etapas para garantir compatibilidade e segurança. Cada etapa representa uma configuração específica que deve ser implantada em todos os seus aplicativos antes de prosseguir para a próxima etapa.

1. Etapa 0: Cliente V2 (ponto de partida) - Sua implementação V2 existente
2. Etapa 1: V4 com compatibilidade V2 - Migre para o cliente V4 enquanto mantém o comportamento de criptografia compatível com V2
3. Etapa 2: V4 com gravações de compromisso de chave - Comece a criptografar com compromisso de chave enquanto permite a descriptografia de objetos legados
4. Etapa 3: V4 com aplicação total - Exigir compromisso de chave para criptografia e decodificação

Etapa 0: Cliente V2 (ponto de partida)

Isso representa sua configuração de cliente V2 existente. Essa etapa demonstra o estado inicial antes da migração.

```
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
});
```

Etapa 1: V4 com compatibilidade com V2

Migre para o cliente V4 enquanto mantém um comportamento idêntico ao V2. Essa etapa usa a `FORBID_ENCRYPT_ALLOW_DECRYPT` política para criptografar sem compromisso e permitir a descriptografia de todos os objetos.

```
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
});
```

Comportamento: criptografa sem compromisso, pode descriptografar objetos comprometidos e não comprometidos. Idêntico ao comportamento da V2.

Etapa 2: V4 com gravações de compromisso importantes

Comece a criptografar com o compromisso de chave e, ao mesmo tempo, mantenha a compatibilidade com versões anteriores para decodificação. Essa etapa usa a `REQUIRE_ENCRYPT_ALLOW_DECRYPT` política.

Warning

Antes de implantar a Etapa 2, certifique-se de que todos os leitores tenham sido atualizados para a Etapa 1 ou posterior para lidar com a criptografia de compromisso de chave.

```
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
});
```

Comportamento: criptografa com comprometimento, pode descriptografar objetos comprometidos e não comprometidos. Novos objetos são protegidos contra ataques de substituição de chaves.

Etapa 3: V4 com aplicação total

Exija o comprometimento da chave tanto para criptografia quanto para decodificação. Essa etapa usa a REQUIRE_ENCRYPT_REQUIRE_DECRYPT política para máxima segurança.

Warning

Antes de implantar a Etapa 3, certifique-se de que todos os objetos em seu sistema tenham sido criptografados novamente com o comprometimento da chave (Etapa 2). Essa etapa falhará ao descriptografar objetos criptografados sem compromisso.

```
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
});
```

Comportamento: criptografa com compromisso, somente descriptografa objetos criptografados com compromisso. Segurança máxima contra ataques de substituição de chaves.

Exemplos adicionais de configuração

Esta seção fornece exemplos adicionais para configurar clientes V4 com opções diferentes durante a migração.

Habilitando o Legacy Support

Para permitir que clientes V4 leiam objetos criptografados por clientes V1 e V2, configure o cliente com uma política de compromisso que permita a descriptografia legada:

```
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
```

Use essa configuração durante a migração quando precisar descriptografar objetos criptografados por clientes mais antigos e, ao mesmo tempo, garantir que novos objetos sejam criptografados com segurança aprimorada.

Configurando o método de armazenamento

A V4 oferece suporte a dois métodos de armazenamento para metadados de criptografia. Escolha o método mais adequado ao seu caso de uso:

Metadados do objeto (padrão)

```
var configuration = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Arquivo de instruções

```
var configuration = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

Use `InstructionFile` quando precisar preservar os metadados do objeto para outras finalidades ou ao trabalhar com objetos que tenham limitações de tamanho de metadados.

Configurando a política de compromisso

Escolha a política de compromisso apropriada com base em seus requisitos de segurança e fase de migração:

Fase de migração (compatibilidade V2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT);
```

Fase de transição (recomendada)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT);
```

Fase de segurança completa

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new
    AmazonS3CryptoConfigurationV4(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT);
```

Comece `FORBID_ENCRYPT_ALLOW_DECRYPT` durante a migração inicial, passe `REQUIRE_ENCRYPT_ALLOW_DECRYPT` para a fase de transição e, finalmente, use `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` quando todos os clientes tiverem sido atualizados e todos os objetos tiverem sido criptografados novamente com compromisso.

Considerações especiais sobre o AWS SDK para .NET

Esta seção contém considerações para casos especiais em que as configurações ou procedimentos normais não são adequados ou suficientes.

Tópicos

- [Como obter assemblies para o AWS SDK para .NET](#)
- [Acessar credenciais e perfis em um aplicativo](#)
- [Considerações especiais sobre o suporte ao Unity](#)
- [Considerações especiais sobre o suporte do Xamarin](#)

Como obter assemblies para o AWS SDK para .NET

Este tópico descreve como você pode obter os assemblies do AWSSDK e armazená-los localmente (ou on-premises) para usá-los em seus projetos. Esse não é o método recomendado para lidar com referências do SDK, mas é obrigatório em alguns ambientes.

Note

O método recomendado para lidar com referências do SDK é baixar e instalar apenas os pacotes NuGet necessários para cada projeto. Esse método é descrito em [Instale pacotes do AWSSDK com o NuGet](#).

Se você não pode ou não tem permissão para baixar e instalar pacotes NuGet por projeto, as seguintes opções estão disponíveis para você.

Faça download e extraia os arquivos ZIP

(Lembre-se de que esse não é o [método recomendado](#) para lidar com referências ao AWS SDK para .NET.)

1. Faça download de um dos arquivos ZIP a seguir:

- [aws-sdk-net8.0.zip](#): assemblies compatíveis com o .NET Core 8 e posterior.

- [aws-sdk-netcoreapp3.1.zip](#) - Assemblies que oferecem suporte para o .NET Core 3.1 e versões posteriores.
- [aws-sdk-netstandard2.0.zip](#) - Assemblies que oferecem suporte para o .NET Standard 2.0 e 2.1.
- [aws-sdk-net45.zip](#) - Assemblies que oferecem suporte para o .NET Framework 4.5 e versões posteriores.
- [aws-sdk-net35.zip](#) - Assemblies que oferecem suporte para o .NET Framework 3.5.

Warning

A partir de 15 de agosto de 2024, o AWS SDK para .NET encerrará o suporte para o .NET Framework 3.5 e a versão mínima do .NET Framework será alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog [Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET](#).

2. Extraia os assemblies em alguma pasta de “download” em seu sistema de arquivos; não importa onde. Anote qual é essa pasta.
3. Ao configurar seu projeto, você obterá os assemblies necessários a partir dessa pasta, conforme descrito em [Instale assemblies do AWSSDK sem o NuGet](#).

Acessar credenciais e perfis em um aplicativo

O método preferido para usar credenciais é permitir que o AWS SDK para .NET encontre e recupere para você, conforme descrito em [Resolução de perfil e credenciais](#)

No entanto, você também pode configurar seu aplicativo para recuperar ativamente perfis e credenciais e, em seguida, usar essas credenciais explicitamente ao criar um cliente de serviço. AWS

[Para recuperar ativamente perfis e credenciais, use classes do Amazon.Runtime.CredentialManagement namespace.](#)

- Para encontrar um perfil em um arquivo que usa o formato de arquivo de AWS credenciais (o arquivo de [AWS credenciais compartilhado em seu local padrão ou um arquivo](#) de credenciais personalizado), use a classe [SharedCredentialsFile](#). Para resumir, algumas vezes, arquivos nesse formato são chamados simplesmente de arquivos de credenciais neste texto.

- Para encontrar um perfil no SDK Store, use a classe [Net SDKCredentials File](#).
- Para pesquisar em um arquivo de credenciais e no SDK Store, dependendo da configuração de uma propriedade de classe, use a [CredentialProfileStoreChain](#) classe.

Você pode usar essa classe para encontrar perfis. Você também pode usar essa classe para solicitar AWS credenciais diretamente em vez de usar a `AWSCredentialsFactory` classe (descrita a seguir).

- Para recuperar ou criar vários tipos de credenciais de um perfil, use a classe [AWSCredentialsFactory](#).

As seções a seguir fornecem exemplos dessas classes.

Exemplos para a aula CredentialProfileStoreChain

Você pode obter credenciais ou perfis da [CredentialProfileStoreChain](#) turma usando os [TryGetProfile](#) métodos [TryGetAWSCredentials](#) ou. A propriedade `ProfilesLocation` da classe determina o comportamento dos métodos, da seguinte forma:

- Se `ProfilesLocation` for nulo ou vazio, pesquise no SDK Store se a plataforma for compatível e, em seguida, pesquise o arquivo de AWS credenciais compartilhado no local padrão.
- Se a propriedade `ProfilesLocation` contiver um valor, pesquise o arquivo de credenciais especificado na propriedade.

Obtenha credenciais do SDK Store ou do arquivo de credenciais compartilhado AWS

Este exemplo mostra como obter credenciais usando a classe `CredentialProfileStoreChain` e, em seguida, use as credenciais para criar um objeto [AmazonS3Client](#). As credenciais podem vir da SDK Store ou do arquivo de credenciais compartilhado da AWS no local padrão.

Este exemplo também usa o [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}");
    }
}
```

Obtenha um perfil na SDK Store ou no arquivo de AWS credenciais compartilhado

Este exemplo mostra como obter um perfil usando a `CredentialProfileStoreChain` classe. As credenciais podem vir do SDK Store ou do arquivo de AWS credenciais compartilhado no local padrão.

Este exemplo também usa a [CredentialProfile](#) classe.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Obtenha credenciais de um arquivo de credenciais personalizado

Este exemplo mostra como obter credenciais usando a `CredentialProfileStoreChain` classe. As credenciais vêm de um arquivo que usa o formato de arquivo de AWS credenciais, mas está em um local alternativo.

Este exemplo também usa o [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
}
```

Exemplos de aulas SharedCredentialsFile e AWSCredentials fábrica

Crie um AmazonS3Client usando a classe SharedCredentialsFile

[Este exemplo mostra como encontrar um perfil no arquivo de AWS credenciais compartilhado, criar credenciais a partir do perfil e, em seguida, usar AWS as credenciais para criar um objeto AmazonS3Client.](#) O exemplo usa a [SharedCredentialsFile](#) classe.

Este exemplo também usa a [CredentialProfile](#) classe e o [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

A classe [Net SDKCredentials File](#) pode ser usada exatamente da mesma forma, exceto que você instanciaria um novo objeto Net SDKCredentials File em vez de um SharedCredentialsFile objeto.

Considerações especiais sobre o suporte ao Unity

Ao usar o AWS SDK para .NET e o [.NET Standard 2.0](#) para seu aplicativo Unity, seu aplicativo deve referenciar os assemblies AWS SDK para .NET (arquivos DLL) diretamente em vez de usar o NuGet. Dado esse requisito, a seguir estão as ações importantes que você precisará realizar.

- Você precisa obter os assemblies do AWS SDK para .NET e aplicá-los ao seu projeto. Para obter informações sobre como fazer isso, consulte [Faça download e extraia os arquivos ZIP](#) no tópico [Como obter assemblies do AWSSDK](#).
- Você precisa incluir as seguintes DLLs em seu projeto Unity junto com as DLLs do AWSSDK.Core e dos outros serviços da AWS que você estiver usando. A partir da versão 3.5.109 do AWS SDK para .NET, o arquivo ZIP padrão .NET contém essas DLLs adicionais.
 - [Microsoft.Bcl.AsyncInterfaces.dll](#)
 - [System.Runtime.CompilerServices.Unsafe.dll](#)
 - [System.Threading.Tasks.Extensions.dll](#)
- Se você estiver usando o [IL2CPP](#) para criar seu projeto Unity, deverá adicionar um arquivo `link.xml` à sua pasta Asset para evitar a remoção do código. O arquivo `link.xml` deve listar todos os assemblies do AWSSDK que você estiver usando e cada um deve incluir o atributo `preserve="all"`. O trecho a seguir mostra um exemplo desse arquivo.

```
<linker>
  <assembly fullname="AWSSDK.Core" preserve="all"/>
  <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
  <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

Para ler informações básicas interessantes relacionadas a esse requisito, consulte o artigo em <https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/>.

Além dessas considerações especiais, consulte [O que mudou na versão 3.5](#) para obter informações sobre como migrar seu aplicativo Unity para a versão 3.5 do AWS SDK para .NET.

Considerações especiais sobre o suporte do Xamarin

Os projetos Xamarin (novos e existentes) devem ter como destino o .NET Standard 2.0. Consulte [Suporte do .NET Standard 2.0 no Xamarin.Forms](#) e [Suporte à implementação do .NET](#).

Veja também as informações sobre a [Portable Class Library e o Xamarin](#).

Referência de API para o AWS SDK para .NET

O AWS SDK para .NET fornece uma API que você pode usar para acessar serviços da AWS. Para ver quais classes e métodos estão disponíveis na API, consulte a [Referência de API do AWS SDK para .NET](#).

Além da referência geral fornecida acima, cada um dos exemplos na seção [Exemplos de código com orientação](#) contém referências aos métodos e classes específicos que são usados nesse exemplo.

Sobre as versões de referência da API

A referência da API descrita anteriormente é somente para a versão 3.0 ou posterior do AWS SDK para .NET.

Para acessar mais informações sobre como migrar de versões mais antigas do SDK, consulte [Migre o seu projeto](#).

Para encontrar conteúdo obsoleto de versões anteriores da API do SDK, consulte os seguintes itens:

- [Referência de API do AWS SDK para .NET V1 \(obsoleta\)](#)
- [Referência de API do AWS SDK para .NET V2 \(obsoleta\)](#)

Para visualizar uma referência de API do AWS SDK para .NET obsoleta, você precisará extraí-la e configurar um navegador da Web. As instruções mostradas a seguir são exemplos de como fazer isso. Elas são baseadas no uso do navegador Google Chrome em um sistema Windows. Adapte-as ao seu navegador e sistema operacional específicos.

Visualize a referência de API V1 obsoleta

1. Baixe o arquivo ZIP da referência de API do AWS SDK para .NET V1 obsoleta. Por padrão, o nome do arquivo ZIP é `sdkfornet-api-ref_v1_deprecated.zip`. Esse nome será usado em todas estas instruções.
2. Coloque o arquivo ZIP em uma pasta de sua escolha. Para essas instruções, presume-se que o nome da pasta seja `C:\work\temp\api-refs\V1`.
3. Clique com o botão direito no arquivo ZIP e escolha Extrair tudo. Aceite o local padrão, que é `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` para essas instruções.

4. Crie um atalho para o Google Chrome na pasta C:\work\temp\api-refs\V1. Tenha cuidado para não mover a aplicação original.
5. Nas propriedades do novo atalho, defina os seguintes campos:
 - Target: "*<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"

Para o argumento --user-data-dir, use um nome de pasta que funcione para o seu ambiente. A pasta não precisa existir.

 - Iniciar no: C:\work\temp\api-refs\V1
6. Forneça ao atalho um nome apropriado.
7. Abra o atalho para ver a referência de API antiga.

Visualizar a referência de API V2 obsoleta

1. Baixe o arquivo ZIP da referência de API do AWS SDK para .NET V2 obsoleta. Por padrão, o nome do arquivo ZIP é sdkfornet-api-ref_v2_deprecated.zip. Esse nome será usado em todas estas instruções.
2. Coloque o arquivo ZIP em uma pasta de sua escolha. Para essas instruções, presume-se que o nome da pasta seja C:\work\temp\api-refs\V2.
3. Clique com o botão direito no arquivo ZIP e escolha Extrair tudo. Aceite o local padrão, que é C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2 para essas instruções.
4. Crie um atalho para o Google Chrome na pasta C:\work\temp\api-refs\V2. Tenha cuidado para não mover a aplicação original.
5. Nas propriedades do novo atalho, defina os seguintes campos:
 - Target: "*<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"

Para o argumento --user-data-dir, use um nome de pasta que funcione para o seu ambiente. A pasta não precisa existir.

 - Iniciar no: C:\work\temp\api-refs\V2
6. Forneça ao atalho um nome apropriado.

7. Abra o atalho para ver a referência de API antiga.

Histórico do documento

A tabela a seguir descreve as alterações importantes feitas desde a versão mais recente do Guia do desenvolvedor do AWS SDK para .NET . Para receber notificações sobre atualizações dessa documentação, inscreva-se em um [feed RSS](#).

Alteração	Descrição	Data
O que há de novo	Foi adicionada uma observação sobre a versão 2.0 da Ferramenta de AWS Implantação para CLI do.NET.	17 de outubro de 2025
AWS OpsWorks	AWS OpsWorks chegou ao fim da vida útil. Foi removido da documentação.	16 de setembro de 2025
O que há de novo	End-of-support foi anunciado para a V3 do AWS SDK para .NET.	21 de agosto de 2025
Estrutura de processamento de mensagens	O AWS Message Processing Framework para.NET atingiu a disponibilidade geral!	5 de maio de 2025
O que há de novo	A versão 4 do AWS SDK para .NET está geralmente disponível! Para acessar mais informações, consulte o Guia do desenvolvedor do AWS SDK para .NET (V4) , especialmente o tópico sobre migração para a versão 4 .	28 de abril de 2025
Novas tentativas e tempos limite	Informações incluídas sobre tempos limite por método que usam CancellationToken	9 de abril de 2025

para chamadas de métodos assíncronas.

[Integração AWS com o.NET Aspire no AWS SDK para .NET](#)

Incluiu informações sobre como AWS foi integrado com o.NET Aspire no. AWS SDK para .NET

15 de fevereiro de 2025

[Observabilidade](#)

Anúncio da versão de disponibilidade geral da observabilidade

10 de fevereiro de 2025

[O que há de novo](#)

Foram adicionadas informações sobre o novo comportamento padrão para proteção de integridade.

15 de janeiro de 2025

[Migração do cliente de criptografia S3 V4](#)

Foram adicionadas informações sobre a migração da V2 para a V4.

9 de dezembro de 2024

[O que há de novo](#)

Foram adicionadas informações sobre a quarta versão prévia da AWS SDK para .NET versão 4.

15 de novembro de 2024

[Observabilidade](#)

Foram adicionadas informações de visualização sobre observabilidade no AWS SDK para .NET, o que permite a coleta de dados de telemetria.

13 de setembro de 2024

Referência de API para o AWS SDK para .NET	As referências de AWS SDK para .NET API para V1 e V2 foram descontinuadas. Foram incluídas informações sobre essa descontinuação e sobre como acessar e visualizar as referências obsoletas.	4 de setembro de 2024
O que há de novo	Foram adicionadas informações sobre a primeira versão prévia da AWS SDK para .NET versão 4.	16 de agosto de 2024
O que há de novo	Foram atualizadas as informações sobre futuras alterações no suporte do .NET Framework.	20 de junho de 2024
O que há de novo	Foram adicionadas informações sobre a versão prévia do AWS Message Processing Framework para .NET	28 de março de 2024
O que há de novo	Foram incluídas informações sobre suporte para o .NET 8.	23 de fevereiro de 2024
O que há de novo	Foram incluídas informações sobre futuras alterações no suporte do .NET Framework.	18 de fevereiro de 2024
Obtenção de AWSSDK montagens	Informações incluídas sobre assemblies compatíveis com o .NET 8 e posterior.	8 de janeiro de 2024
AWS Estrutura de processamento de mensagens para .NET	Foram incluídas informações sobre a versão Beta do Message Processing Framework.	10 de dezembro de 2023

[Como usar os bancos de dados NoSQL do Amazon DynamoDB](#)

Informações atualizadas sobre os modelos de programação de persistência de documentos e objetos. Agora é possível evitar certas condições de latência ou deadlock devido aos comportamentos de inicialização a frio e de pool de threads.

15 de novembro de 2023

[Inclusão de mais atualizações de práticas recomendadas do IAM](#)

Guia atualizado para alinhamento com as práticas recomendadas do IAM. Para obter mais informações, consulte [Práticas recomendadas de segurança no IAM](#).

5 de outubro de 2023

[Obtenção de AWSSDK montagens](#)

Foram removidas as informações sobre a instalação do AWS SDK para .NET usando o AWS Tools para Windows instalador (ou seja, o MSI), que se tornou obsoleto.

25 de setembro de 2023

[Atualizações de práticas recomendadas do IAM](#)

Guia atualizado para alinhamento com as práticas recomendadas do IAM. Para obter mais informações, consulte [Práticas recomendadas de segurança no IAM](#).

18 de julho de 2023

[Lambda Annotations](#)

A estrutura AWS Lambda Annotations foi lançada para disponibilidade geral.

17 de julho de 2023

O que há de novo	Foram adicionadas informações sobre a versão prévia do Distributed Cache Provider for DynamoDB.	15 de julho de 2023
Índice	Índice atualizado para tornar os exemplos de código mais facilmente detectáveis.	8 de junho de 2023
Resolução da região	Foram adicionadas informações sobre como o SDK resolve uma especificação de região ausente.	14 de março de 2023
Suporte para o MSI	Foi adicionada uma observação sobre o fim do suporte para o AWS Tools para Windows instalador.	6 de março de 2023
Lambda Annotations (Pré-visualização)	Pré-visualização da estrutura de AWS Lambda anotações.	22 de setembro de 2022
Implante aplicativos em AWS	Conteúdo principal movido para um site do GitHub Pages: https://aws.github.io/aws-dotnet-deploy/	28 de junho de 2022
Aposentadoria - Clássico EC2	Foram adicionadas notas sobre a aposentadoria do EC2 -Classic.	13 de abril de 2022
Login único com o AWS SDK para .NET	Foram adicionadas informações sobre a autenticação única (SSO) ao usar o AWS SDK para .NET.	17 de março de 2022
Aplicar uma versão mínima do TLS	Foram adicionadas informações sobre TLS 1.3.	16 de março de 2022

Trabalhe com AWS serviços	Listas incluídas dos exemplos de código que estão disponíveis em GitHub.	28 de fevereiro de 2022
Como habilitar as métricas do SDK	As informações sobre a habilitação de métricas do SDK, que foi descontinuado, foram removidas.	20 de janeiro de 2022
Implante aplicativos em AWS	Foi adicionada uma referência ao AWS Toolkit for Visual Studio, que fornece funcionalidade de implantação semelhante à AWS Ferramenta de Implantação.	26 de outubro de 2021
AWS SDK para .NET consolidação do guia da versão 3	Os dois guias do desenvolvedor da AWS SDK para .NET versão 3, "V3" e "mais recente", foram consolidados em um guia sob o URL "v3".	18 de agosto de 2021
Migração do .NET Standard 1.3	O suporte do .NET Standard 1.3 no AWS SDK para .NET chegou ao fim de sua vida útil.	25 de março de 2021
Implante aplicativos em AWS (versão prévia)	Foram adicionadas informações de visualização sobre a Ferramenta de AWS Implantação, que você pode usar para implantar um aplicativo a partir da CLI do .NET.	15 de março de 2021
Versão 3.5 do AWS SDK para .NET	A versão 3.5 do AWS SDK para .NET foi lançada.	25 de agosto de 2020

<u>Paginadores</u>	Foram adicionados paginadores a muitos clientes de serviços, o que torna a paginação dos resultados da API mais conveniente.	24 de agosto de 2020
<u>Novas tentativas e tempos limite</u>	Foram adicionadas informações sobre os modos de novas tentativas.	20 de agosto de 2020
<u>Usar chaves do KMS para a criptografia do S3</u>	Exemplo atualizado para usar a versão 2 para a criptografia do S3 client.	6 de agosto de 2020
<u>Migração do .NET Standard 1.3</u>	Adicionadas informações sobre como terminar o suporte para o .NET Standard 1.3 no final de 2020.	18 de maio de 2020
<u>Início rápido</u>	Adição de uma seção de início rápido com configuração básica e tutoriais para apresentar o leitor ao AWS SDK para .NET.	27 de março de 2020
<u>Impor o TLS 1.2</u>	Foram adicionadas informações sobre como impor o TLS 1.2 no SDK.	10 de março de 2020