



Guía para desarrolladores

AWS SDK para .NET (V4)



AWS SDK para .NET (V4): Guía para desarrolladores

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

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

.....	x
¿Qué es el AWS SDK para .NET	1
Acerca de esta versión	1
Mantenimiento y compatibilidad de las versiones principales del SDK	2
Casos de uso comunes	2
Temas adicionales en esta sección	3
AWS Herramientas relacionadas	3
Herramientas para Windows PowerShell y herramientas para PowerShell Core	3
Kit de herramientas para VS Code	3
Kit de herramientas para Visual Studio	4
Kit de herramientas para Azure DevOps	4
Plataformas admitidas	4
.NET Core	4
.NET Standard 2.0	5
.NET Framework 4.5	5
.NET Framework 3.5	5
Biblioteca de clases portátil y Xamarin	5
Compatibilidad con Unity	5
SDKs y referencia de herramientas	5
Historial de revisiones	6
Novedades	6
Recursos adicionales	8
Introducción	11
Instalación y configuración de su cadena de herramientas	11
Desarrollo multiplataforma	11
Windows con Visual Studio y .NET Core	12
Siguiente paso	12
Autenticarse con AWS	12
Inicie sesión con las credenciales de la consola	13
Habilitar y configurar el Centro de identidades de IAM	14
Configuración de SDK para usar IAM Identity Center	14
Inicie una sesión en el portal de AWS acceso	16
Información adicional	16
Crear una aplicación sencilla	17

Aplicación multiplataforma sencilla	18
Aplicación sencilla basada en Windows	23
Pasos a seguir a continuación	29
Configuración	30
Inicio de un nuevo proyecto	30
Instale AWSSDK paquetes con NuGet	32
NuGet Utilizándolo desde la línea de comandos o el terminal	33
NuGet Utilizándolo desde el Explorador de soluciones de Visual Studio	33
Uso NuGet desde la consola de Package Manager	34
Instale AWSSDK conjuntos sin NuGet	35
AWS Región	36
Creación de un cliente de servicio con una región particular	37
Especificación de una región para todos los clientes de servicio	38
Resolución de la región	39
Información especial sobre la región de China (Pekín)	39
Información especial sobre nuevos AWS servicios	39
Resolución de credencial y perfil	40
Resolución de perfiles	41
Uso de credenciales de cuentas de usuario federado	41
Especificación de roles o credenciales temporales	42
Uso de credenciales de proxy	42
Reintentos y tiempos de espera	43
Reintentos	44
Tiempos de espera	45
Observabilidad	48
Recursos adicionales	48
Configuración de un TelemetryProvider	48
Métricas	50
Proveedores de telemetría	52
Usuarios y roles	54
Usuarios y conjuntos de permisos	54
Roles de servicio	54
Configuración avanzada	55
AWSSDK.Extensions. NETCore.Configuración y IConfiguration	56
Configuración de otros parámetros de la aplicación	60
Referencia de archivos de configuración para AWS SDK para .NET	68

Uso de credenciales heredadas	81
Advertencias y directrices importantes para las credenciales	82
Uso del archivo de AWS credenciales compartido	83
Uso de SDK Store (solo Windows)	86
Uso del SDK	91
Programación asíncrona	91
Paginación	93
¿Dónde puedo encontrar paginadores?	93
¿Qué me proporcionan los paginadores?	93
Paginación sincrónica frente a paginación asincrónica	94
Ejemplo	94
Consideraciones adicionales sobre los paginadores	98
Support para HTTP 2	99
Consideraciones adicionales	101
Herramientas adicionales	101
AWS Herramienta de implementación	102
AWS Marco de procesamiento de mensajes para .NET	102
Integraciones con .NET Aspire	102
Autenticación avanzada	103
Inicio de sesión único	103
Requisitos previos	104
Configuración de un perfil de SSO	104
Generación y uso de tokens de SSO	106
Recursos adicionales	111
Tutoriales	111
Tutorial: solo aplicación .NET	111
Tutorial: AWS CLI y aplicación.NET	121
Implemente en AWS	130
Implementación desde la CLI de .NET	130
Implementación desde kits de herramientas de IDE	130
Casos de uso	131
Aplicaciones ASP.NET Core	131
Aplicaciones de consola de .NET	132
Aplicaciones Blazor WebAssembly	133
AWS Lambda proyectos	134
Requisitos previos	134

Comandos de Lambda disponibles	135
Pasos de la implementación	135
AWS Servicios de llamadas	137
Ejemplos de código guiados	137
CloudFormation	138
Amazon Cognito	142
DynamoDB	151
Amazon EC2	181
IAM	244
Amazon S3	263
Amazon SNS	274
Amazon SQS	278
AWS Lambda	312
APIs	312
Requisitos previos	312
Información adicional	312
Temas	312
Lambda Annotations	312
Bibliotecas y marcos de trabajo de alto nivel	314
Marco de procesamiento de mensajes	315
Integración AWS con .NET Aspire	335
Otros servicios y configuraciones	336
Ejemplos de código	338
Aurora	339
Introducción	339
Conceptos básicos	341
Acciones	366
escalado automático	381
Introducción	339
Conceptos básicos	341
Acciones	366
Amazon Bedrock	417
Introducción	339
Acciones	366
Amazon Bedrock Runtime	421
Anthropic Claude	421

Cohere Command	427
Meta Llama	431
Mistral AI	434
CloudFormation	438
Introducción	339
CloudWatch	442
Introducción	339
Conceptos básicos	341
Acciones	366
CloudWatch Registros	500
Acciones	366
Escenarios	504
Amazon Cognito Identity Provider	526
Acciones	366
AWS Control Tower	527
Introducción	339
Conceptos básicos	341
Acciones	366
DynamoDB	567
Introducción	339
Conceptos básicos	341
Acciones	366
Amazon EC2	607
Introducción	339
Amazon ECS	609
Introducción	339
AWS IoT	612
Introducción	339
Conceptos básicos	341
Acciones	366
AWS IoT data	671
Acciones	366
Amazon Redshift	674
Introducción	339
Conceptos básicos	341
Acciones	366

Amazon S3	705
Introducción	339
Conceptos básicos	341
Acciones	366
Escenarios	504
Migración de su proyecto	742
Migración a la versión 4	742
.NET Framework	742
Tipos de valor	743
Colecciones	743
AWS Security Token Service (STS)	744
Cambios relacionados con ClientConfig	744
Las AWSSDK.Extensions.NETCoreEl paquete .Setup NuGet	745
CookieSigner y UrlSigner	746
DateTime frente a UTC DateTime	746
DateTime análisis	747
ConvertFromUnixEpochSeconds y ConvertFromUnixEpochMilliseconds	747
Registro	747
Support para HTTP 2	748
Inicio de sesión único	748
Cambios específicos de DynamoDB	749
Cambios específicos de EC2	752
Cambios específicos de S3	753
Elementos de programación que se eliminaron	757
Migración desde .NET Standard 1.3	760
Seguridad	762
Protección de datos	763
Gestión de identidad y acceso	764
Público	764
Autenticación con identidades	765
Administración del acceso con políticas	766
¿Cómo Servicios de AWS trabajar con IAM	768
Solución de problemas de AWS identidad y acceso	768
Validación de la conformidad	770
Resiliencia	771
Seguridad de infraestructuras	771

Aplicación de una versión mínima de TLS	772
.NET Core	772
.NET Framework	773
Herramientas de AWS para PowerShell	774
Xamarin	775
Unity	776
Navegador (para Blazor) WebAssembly	776
Migración del cliente de cifrado S3 (V1 a V2)	777
Información general sobre la migración	777
Actualización de los clientes existentes a clientes de transición a la versión 1 para leer nuevos formatos	778
Migración de los clientes de transición a la versión 1 a clientes de la versión 2 para escribir nuevos formatos	779
Actualización de clientes de la versión 2 para que dejen de leer formatos de la versión 1	782
Migración del cliente de cifrado S3 (de V2 a V4)	782
Comprender los conceptos de la versión 4	783
Actualice los clientes existentes para leer los formatos V4	784
Migre a clientes V4	785
Ejemplos de configuración adicionales	789
Consideraciones especiales	792
Obtención de AWSSDK conjuntos	792
Descarga y extracción de archivos ZIP	792
Acceso a las credenciales y perfiles en una aplicación	793
Ejemplos de clase CredentialProfileStoreChain	794
Ejemplos de clases SharedCredentialsFile y AWSCredentials Factory	795
Compatibilidad con Unity	796
Compatibilidad con Xamarin	797
Referencia de la API	798
Acerca de las versiones de referencia de la API	798
Historial de documentos	801

¡Se AWS SDK para .NET ha publicado la versión 4 (V4) del!

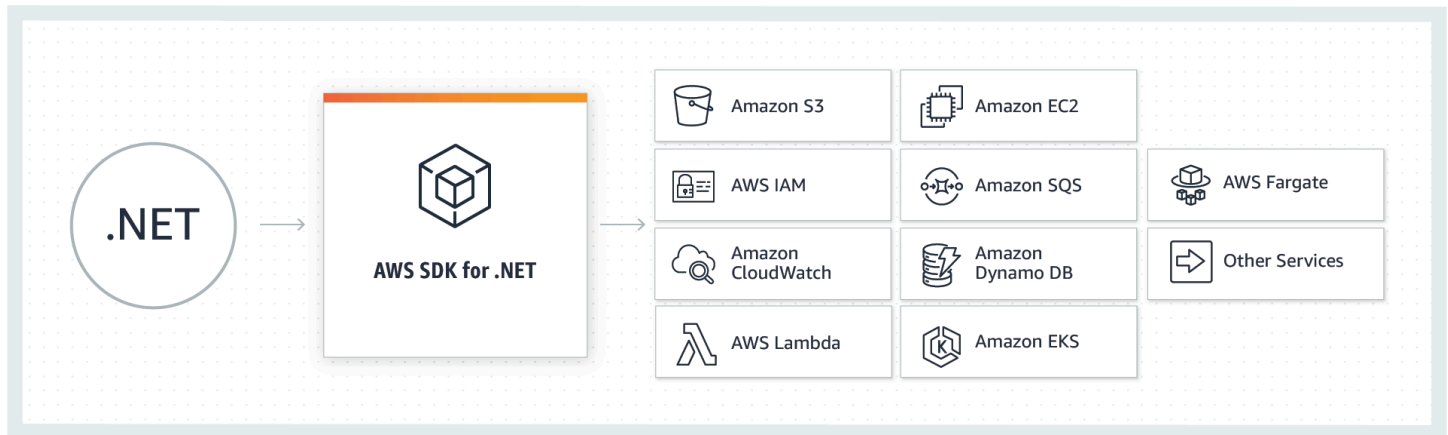
Para obtener información sobre los cambios más importantes y la migración de sus aplicaciones, consulte el [tema sobre migración](#).



Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.

¿Qué es el AWS SDK para .NET

AWS SDK para .NET Esto facilita la creación de aplicaciones.NET que aprovechan AWS servicios rentables, escalables y confiables, como Amazon Simple Storage Service (Amazon S3) y Amazon Elastic Compute Cloud (Amazon EC2). SDK simplifica el uso de servicios de AWS , ya que proporciona un conjunto de bibliotecas uniforme con las los desarrolladores de .NET están familiarizados.



(¡Vale, lo tengo! Estoy listo para [prepararme](#) y [hacer un recorrido rápido](#)).

Acerca de esta versión

Documentación actual y pasada

Esta documentación corresponde a la versión 4.0 y posteriores (V4) del AWS SDK para .NET. Gira en torno a .NET Core y ASP.NET Core principalmente, pero también contiene información sobre .NET Framework y ASP.NET 4 x. Además de Windows y Visual Studio, analiza de igual forma el desarrollo multiplataforma.

Para obtener información sobre la migración desde versiones anteriores del SDK, consulte [Migración de su proyecto](#).

Para obtener información sobre la versión anterior del SDK, la V3, que quedará obsoleta en el futuro, consulta la Guía para desarrolladores [AWS SDK para .NET \(V3\)](#).

Para encontrar contenido obsoleto de versiones anteriores del SDK, consulta los siguientes elementos:

- [AWS SDK para .NET \(versión 2, obsoleta\) Guía para desarrolladores](#)

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

Mantenimiento y compatibilidad de las versiones principales del SDK

Para obtener información sobre el mantenimiento y el soporte de las versiones principales del SDK y sus dependencias subyacentes, consulta lo siguiente en la Guía de [referencia de herramientas AWS SDKs y herramientas](#):

- [AWS SDKs y política de mantenimiento de herramientas](#)
- [AWS SDKs matriz de soporte de versiones y herramientas](#)

Casos de uso comunes

AWS SDK para .NET Esto le ayuda a darse cuenta de varios casos de uso convincentes, incluidos los siguientes:

- Administración de usuarios y roles con [AWS Identity and Access Management \(IAM\)](#)
- Acceso a [Amazon Simple Storage Service \(Amazon S3\)](#) para crear buckets y almacenar objetos
- Administración de suscripciones HTTP a temas de [Amazon Simple Notification Service \(Amazon SNS\)](#)
- Uso de la [Utilidad de transferencias de S3](#) para transferir archivos a Amazon S3 desde aplicaciones Xamarin
- Uso de [Amazon Simple Queue Service \(Amazon SQS\)](#) para procesar mensajes y flujos de trabajo entre componentes de un sistema.
- Realización de transferencias de Amazon S3 eficaces mediante el envío de instrucciones SQL a [Amazon S3 Select](#)
- Cree y lance EC2 instancias de [Amazon](#), y configure y solicite [instancias EC2 puntuales](#) de Amazon.

Temas adicionales en esta sección

- [AWS herramientas relacionadas con AWS SDK para .NET](#)
- [Plataformas compatibles con AWS SDK para .NET](#)
- [AWS SDKs Guía de referencia de herramientas y herramientas](#)
- [Historial de revisiones](#)
- [Qué hay de nuevo en el AWS SDK para .NET](#)
- [Recursos adicionales](#)

AWS herramientas relacionadas con AWS SDK para .NET

Herramientas para Windows PowerShell y herramientas para PowerShell Core

Los AWS Tools for Windows PowerShell y AWS Tools for PowerShell Core son PowerShell módulos que se basan en la funcionalidad expuesta por el AWS SDK para .NET. Las AWS PowerShell herramientas le permiten programar las operaciones de sus AWS recursos a partir del PowerShell indicador. Si bien los cmdlets se implementan mediante los clientes de servicio y los métodos del SDK, proporcionan una PowerShell experiencia idiomática para especificar parámetros y gestionar los resultados.

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

Kit de herramientas para VS Code

El [AWS Toolkit for Visual Studio Code](#) es un complemento para el editor de código de Visual Studio (VS Code). El conjunto de herramientas facilita el desarrollo, la depuración y la implementación de aplicaciones que utilizan AWS.

Con el conjunto de herramientas, puede hacer cosas como las siguientes:

- Cree aplicaciones sin servidor que contengan AWS Lambda funciones y, a continuación, impleméntelas en una pila. AWS CloudFormation
- Trabaja con los EventBridge esquemas de Amazon.
- Úselo IntelliSense cuando trabaje con archivos de definición de tareas de Amazon ECS.

- Visualice una aplicación AWS Cloud Development Kit (AWS CDK) .

Kit de herramientas para Visual Studio

AWS Toolkit for Visual Studio Es un complemento para el IDE de Visual Studio que facilita el desarrollo, la depuración y la implementación de aplicaciones.NET que utilizan Amazon Web Services. Kit de herramientas para Visual Studio proporciona plantillas de Visual Studio para servicios como Lambda, además de asistentes de implementación para aplicaciones web y aplicaciones sin servidor. Puede utilizar el AWS explorador para gestionar EC2 instancias de Amazon, trabajar con tablas de Amazon DynamoDB, publicar mensajes en las colas de Amazon Simple Notification Service (Amazon SNS) y mucho más, todo desde Visual Studio.

Para empezar, consulte [Configuración del AWS Toolkit for Visual Studio](#).

Kit de herramientas para Azure DevOps

Kit de herramientas de AWS para Microsoft Azure DevOps Agrega tareas para permitir fácilmente la creación y el lanzamiento de canalizaciones en Azure DevOps y Azure DevOps Server para que funcionen con AWS los servicios. Puede trabajar con Amazon S3, AWS Elastic Beanstalk, Lambda AWS CodeDeploy AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) y Amazon SNS. También puede ejecutar comandos mediante el PowerShell módulo de Windows y la tecla (). AWS Command Line Interface AWS CLI

Para empezar con Kit de herramientas de AWS para Azure DevOps, consulte la [Guía del Kit de herramientas de AWS para Microsoft Azure DevOps usuario](#).

Plataformas compatibles con AWS SDK para .NET

AWS SDK para .NET Proporciona distintos grupos de ensamblajes para que los desarrolladores se dirijan a diferentes plataformas. Sin embargo, no toda la funcionalidad del SDK es la misma en cada una de estas plataformas. En este tema se describen las diferencias en cuanto a soporte de cada plataforma.

.NET Core

AWS SDK para .NET Es compatible con las aplicaciones escritas para .NET Core (.NET Core 3.1, .NET 5, .NET 6, etc.). AWS los clientes de servicio solo admiten patrones de llamadas asíncronas en .NET core. Esto también afecta a muchas de las abstracciones generales basadas

en clientes de servicio, como la TransferUtility de Amazon S3, que solo admitirá llamadas asincrónicas en el entorno de .NET Core.

.NET Standard 2.0

[Las variantes de los que no son de Framework AWS SDK para .NET cumplen con el estándar 2.0 de .NET](#). AWS SDK para .NET proporciona solo métodos asincrónicos para las aplicaciones escritas con .NET Standard.

.NET Framework 4.5

Esta versión de AWS SDK para .NET está compilada con la versión 4.7.2 de .NET Framework y se ejecuta en el entorno de ejecución .NET 4.0. AWS [los clientes de servicio admiten patrones de llamadas sincrónicas y asíncronas y utilizan las palabras clave async y await introducidas en C# 5.0](#).

.NET Framework 3.5

La versión 4 no es compatible con .NET Framework 3.5 AWS SDK para .NET .

Biblioteca de clases portátil y Xamarin

AWS SDK para .NET También contiene una implementación de biblioteca de clases portátil. La implementación de la biblioteca de clases portátil puede tener como destino varias plataformas, como la Plataforma universal de Windows (UWP) y Xamarin en iOS y Android. Consulte el [SDK móvil para .NET and Xamarin para obtener](#) más información. AWS los clientes de servicio solo admiten patrones de llamadas asíncronas.

Compatibilidad con Unity

Para obtener más información sobre la compatibilidad con Unity, consulte [Consideraciones especiales sobre la compatibilidad con Unity](#).

AWS SDKs Guía de referencia de herramientas y herramientas

La [guía de referencia de AWS SDKs and Tools](#) contiene información relevante e importante para muchos de los kits de herramientas AWS SDKs y para AWS CLI Estos son algunos ejemplos de la información que la guía de referencia contiene:

- Información sobre los [archivos de AWS compartidos config y credentials](#) y su [ubicación](#)

- [Configuración de AWS cuentas, usuarios y roles](#)
- [Referencia de ajustes de configuración y autenticación](#)
- [AWS Bibliotecas de Common Runtime \(CRT\)](#)
- [AWS SDKs política de mantenimiento de herramientas y herramientas](#)
- [AWS SDKs matriz de soporte de versiones y herramientas](#)

Historial de revisiones

Para saber qué ha cambiado en las distintas versiones, consulte lo siguiente:

- [Registros de cambios del SDK](#)
- [Qué hay de nuevo en el AWS SDK para .NET](#)
- [Historial de documentos](#)

Qué hay de nuevo en el AWS SDK para .NET

Para obtener información de alto nivel sobre los nuevos desarrollos relacionados con el AWS SDK para .NET, consulte la página del producto en <https://aws.amazon.com/sdk-for-net/> y los [registros de cambios del SDK](#).

Estas son las novedades de AWS SDK para .NET.

17 de octubre de 2025: versión 2.0 de la herramienta de AWS implementación

Se ha publicado la versión 2.0 de la herramienta de AWS implementación para .NET CLI. Para obtener información acerca de la herramienta de implementación, consulte [Implemente aplicaciones en AWS](#). Para obtener información sobre la versión 2.0 de la herramienta de despliegue, consulte la entrada [del blog Novedades de la herramienta de AWS despliegue para .NET](#).

21 de agosto de 2025: Próximamente end-of-support disponible para la versión 3 de AWS SDK para .NET

Se AWS SDK para .NET ha anunciado la end-of-support versión 3 (V3) del. Consulte la [guía de migración](#) para migrar sus aplicaciones de la V3 y evitar interrupciones. Para obtener más información, consulte la entrada del blog en la que se [anuncia la end-of-support AWS SDK para .NET versión 3](#).

2 de julio de 2025: disponibilidad general del proveedor de caché distribuida

¡El proveedor de caché distribuida AWS de .NET para Amazon DynamoDB ya está disponible de forma general! Este proveedor implementa la interfaz Core [IDistributedCache](#) de ASP.NET, lo que le permite integrar la infraestructura duradera y totalmente gestionada de DynamoDB en la capa de almacenamiento en caché con cambios de código mínimos. El proveedor de caché distribuida está disponible a través de [AWS. AspNetCore. DistributedCacheProvider](#) paquete en NuGet. Para obtener más información, consulte la entrada del blog El [proveedor de caché distribuida AWS .NET para Amazon DynamoDB](#) ya está disponible de forma general.

Note

La versión publicada del proveedor de caché distribuida solo es compatible con la versión 4 del AWS SDK para .NET. Si probó la versión preliminar del proveedor de caché distribuida, es posible que necesite actualizar las dependencias de los paquetes en su aplicación.

5 de mayo de 2025: disponibilidad general del marco de procesamiento de AWS mensajes para .NET

¡El [marco de procesamiento de AWS mensajes para .NET](#) está disponible de forma general! El marco es un marco AWS nativo que simplifica el desarrollo de aplicaciones de procesamiento de mensajes de .NET que utilizan AWS servicios como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) y Amazon EventBridge.

28 de abril de 2025: versión 4 del AWS SDK para .NET

¡La versión 4 del AWS SDK para .NET está disponible de forma general! Para obtener información sobre la migración de sus aplicaciones a la V4, consulte [Migración a la versión 4](#). Consulte también la entrada del blog sobre la [disponibilidad general de la AWS SDK para .NET versión 4.0](#), en la que se anuncia la disponibilidad general.

15 de febrero de 2025: integraciones con .NET Aspire

Se han lanzado las integraciones con .NET Aspire para mejorar el ciclo de desarrollo interno. Para obtener información, consulte [Integración AWS con .NET Aspire en AWS SDK para .NET](#).

10 de febrero de 2025: versión de GA para la observabilidad

La observabilidad es la medida en que se puede deducir el estado actual de un sistema a partir de los datos que emite. Se ha añadido la observabilidad AWS SDK para .NET, incluida la

implementación de un proveedor de telemetría. Para obtener más información, consulte esta guía y la entrada del blog [Observabilidad](#) en la que se [anuncia la disponibilidad general de AWS las bibliotecas.NET. OpenTelemetry](#)

15 de enero de 2025: nuevo comportamiento predeterminado para la protección de la integridad

A partir de la versión 3.7.412.0 del AWS SDK para .NET, el SDK proporciona protecciones de integridad predeterminadas al calcular automáticamente una CRC32 suma de verificación para las cargas. Para obtener más información, consulta el anuncio en. GitHub <https://github.com/aws/aws-sdk-net/issues/3610> El SDK también proporciona una configuración global para las protecciones de la integridad de los datos que puede configurar externamente, sobre las que puede leer en la [Guía de referencia sobre las protecciones de la integridad de los datosAWS SDKs y las herramientas](#).

15 de noviembre de 2024: entrada de blog sobre la versión preliminar 4 de la versión 4

La versión preliminar 4 de la AWS SDK para .NET versión 4 se publicó el 15 de noviembre de 2024. Para obtener más información sobre esta versión preliminar, consulte la entrada del blog [Vista previa 4 de la AWS SDK para .NET versión 4](#).

16 de agosto de 2024: entrada de blog sobre la versión preliminar 1 de la versión 4

La AWS SDK para .NET versión 4 se publicó como primera vista previa el 16 de agosto de 2024. Para obtener más información sobre esta versión preliminar, consulte la entrada del blog [Vista previa 1 de la AWS SDK para .NET versión 4](#).

Recursos adicionales

Servicios admitidos

AWS SDK para .NET Es compatible con la mayoría de los productos de AWS infraestructura y se agregan más servicios con frecuencia. Para obtener una lista de los AWS servicios compatibles con el SDK, consulte el [archivo README del SDK](#).

Página de inicio del AWS SDK para .NET

Para obtener más información sobre el AWS SDK para .NET, consulte la página de inicio del SDK en <https://aws.amazon.com/sdk-for-net/>.

Documentación de referencia de SDK

La documentación de referencia de SDK proporciona la capacidad de explorar y buscar entre todo el código incluido en SDK. Proporciona documentación muy completa y ejemplos de uso. Para obtener más información, consulte la [Referencia de la API de AWS SDK para .NET](#).

AWS Re:post (anteriormente AWS foros)

Visita [AWS Re:post](#), específicamente el [tema del mismo AWS SDK para .NET, para](#) hacer preguntas o enviar comentarios sobre él. AWS Al final de cada página de la documentación hay un enlace para probar AWS re:post que lleva al tema de re:post correspondiente. Los ingenieros de AWS supervisan los temas y responden a las preguntas, comentarios y problemas.

Si ha iniciado sesión en re:post, también puede seguir un tema. Para seguir el tema AWS SDK para .NET, ve a la [página Todos los temas](#), busca «.NET en AWS» y selecciona el botón Seguir.

Kits de herramientas

- AWS Toolkit for Visual Studio: si utiliza el IDE de Microsoft Visual Studio, debe echar un vistazo a la [Guía del usuario del AWS Toolkit for Visual Studio](#).
- AWS Toolkit for Visual Studio Code: si utiliza el IDE de Microsoft Visual Studio, debe echar un vistazo a la [Guía del usuario del AWS Toolkit for Visual Studio Code](#).

Bibliotecas, extensiones y herramientas de utilidad

Visite los aws-sdk-net repositorios [aws/dotnet](#) y [aws/](#) del sitio GitHub web para obtener enlaces a bibliotecas, herramientas y recursos que puede utilizar para crear aplicaciones y servicios de.NET. AWS

A continuación se muestran algunos ejemplos:

- [AWS Extensión de configuración.NET para Systems Manager](#)
- [AWS Extensiones: Configuración de .NET Core](#)
- [AWS Registro de.NET](#)
- [Biblioteca de extensiones de autenticación de Amazon Cognito](#)
- [AWS X-Ray SDK para .NET](#)

Otros recursos

Los siguientes son otros recursos que podrían resultar útiles:

- [.NET para desarrolladores](#)
- [Entorno de desarrollo de .NET en la AWS nube: implementación de referencia de inicio rápido](#)
- [Blog Hello, Cloud!](#)
- AWS Documento técnico: [Desarrollo e implementación de aplicaciones .NET](#) en AWS
- [AWS Microservice Extractor for .NET](#)
- [Asistente de portabilidad para .NET](#)
- [AWS SDKs y guía de referencia de herramientas](#)

Cómo empezar con el AWS SDK para .NET

Aprenda a instalar, configurar y usar el AWS SDK para .NET. Debe instalar su cadena de herramientas y configurar una serie de elementos esenciales que su aplicación necesita para acceder a AWS los servicios. Entre ellos se incluyen:

- Una cuenta de usuario o un rol adecuados
- Información de autenticación de esa cuenta de usuario o para asumir ese rol

Para obtener información sobre cómo configurar los proyectos y otras partes críticas, consulte [Configuración del AWS SDK para .NET](#).

Temas

- [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#)
- [Autenticar el con AWS SDK para .NET AWS](#)
- [Creación de una aplicación sencilla mediante el AWS SDK para .NET](#)

Instalación y configuración de la cadena de herramientas para AWS SDK para .NET

Para utilizarla AWS SDK para .NET, debe tener determinadas herramientas de desarrollo instaladas.

Desarrollo multiplataforma

Lo siguiente es necesario para el desarrollo multiplataforma de .NET en Windows, Linux o macOS:

- Microsoft [.NET Core SDK](#), versión 2.1, 3.1 o posterior, que incluye la interfaz de línea de comandos (CLI) de .NET (**dotnet**) y el tiempo de ejecución de .NET Core.
- Un editor de código o entorno de desarrollo integrado (IDE) adecuado para el sistema operativo y los requisitos. Normalmente, es uno que proporciona cierto soporte para .NET Core.

Algunos ejemplos son [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) y [Microsoft Visual Studio](#).

- (Opcional) Un AWS kit de herramientas, si está disponible, para el editor que elija y su sistema operativo.

Entre los ejemplos se incluyen [AWS Toolkit for Visual Studio Code](#), [AWS Toolkit for JetBrains](#) y [AWS Toolkit for Visual Studio](#).

Windows con Visual Studio y .NET Core

Lo siguiente es necesario para el desarrollo en Windows con Visual Studio y .NET Core:

- [Microsoft Visual Studio](#)

- Microsoft .NET Core 2.1, 3.1 o posterior

Esto normalmente se incluye de forma predeterminada al instalar una versión reciente de Visual Studio.

- (Opcional) The AWS Toolkit for Visual Studio, que es un complemento que proporciona una interfaz de usuario para administrar AWS los recursos y los perfiles locales desde Visual Studio. Para instalar el kit de herramientas, consulte [Configuración del AWS Toolkit for Visual Studio](#).

Para obtener más información, consulte la [Guía del usuario de AWS Toolkit for Visual Studio](#).

Siguiente paso

[Autenticar el con AWS SDK para .NET AWS](#)

Autenticar el con AWS SDK para .NET AWS

Debes establecer cómo se autentica tu código AWS al desarrollar con. Servicios de AWS Existen diferentes formas de configurar el acceso programático a AWS los recursos, según el entorno y el AWS acceso del que dispongas.

Para ver los distintos métodos de autenticación del SDK, consulte [Autenticación y acceso](#) en la Guía de referencia sobre herramientas AWS SDKs y herramientas.

En este tema se presupone que un nuevo usuario se está desarrollando a nivel local, que su empresa no le ha proporcionado un método de autenticación y que lo utilizará AWS IAM Identity

Center para obtener credenciales temporales. Si el entorno no se basa en estos supuestos, es posible que parte de la información de este tema no se aplique a su caso o que ya se le haya proporcionado parte de la información.

La configuración de este entorno requiere varios pasos, que se resumen de la siguiente manera:

1. [Inicie sesión con las credenciales de la consola](#)
2. [Habilitar y configurar el Centro de identidades de IAM](#)
3. [Configuración de SDK para usar IAM Identity Center](#)
4. [Inicie una sesión en el portal de AWS acceso](#)

Inicie sesión con las credenciales de la consola

Puede usar sus credenciales de inicio de sesión AWS de Management Console actuales para acceder a AWS los servicios mediante programación. Tras un flujo de autenticación basado en un navegador, AWS genera credenciales temporales que funcionan en todas las herramientas de desarrollo locales, como la AWS CLI Herramientas de AWS para PowerShell y el AWS SDK para .NET v4.

Con este proceso, puede autenticarse mediante las credenciales raíz creadas durante la configuración inicial de la cuenta, un usuario de IAM o una identidad federada de su proveedor de identidad y la AWS CLI, o bien gestionar Herramientas de AWS para PowerShell automáticamente las credenciales temporales por usted. Este enfoque mejora la seguridad al eliminar la necesidad de almacenar localmente las credenciales a largo plazo. Al ejecutar el `Invoke-AWSLogin` cmdlet Herramientas de AWS para PowerShell o el `aws login` comando desde la AWS CLI, puede seleccionar una de las sesiones de consola activas o iniciar sesión mediante el flujo de autenticación basado en el navegador, lo que generará credenciales temporales automáticamente. El token de credencial temporal caducará en 15 minutos, pero el AWS SDK para .NET v4 actualizará automáticamente el token cuando sea necesario durante tus solicitudes.

Important

Si utilizas las credenciales de la consola para la autenticación, tu aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución de AWS inicio de sesión funcione:

- AWSSDK4. Iniciar sesión

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

- [Herramientas de AWS para PowerShell Guía del desarrollador de credenciales de inicio de sesión.](#)
- [Inicie sesión para el desarrollo AWS local con las credenciales de la consola mediante AWS CLI.](#)

El config archivo tendrá este aspecto una vez que haya iniciado sesión siguiendo los pasos indicados anteriormente para nuestra Herramientas de AWS para PowerShell 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 y configurar el Centro de identidades de IAM

Para usar IAM Identity Center, primero debe estar habilitado y configurado. Para obtener más información sobre cómo hacerlo con el SDK, consulte el paso 1 del tema sobre la [autenticación del Centro de Identidad de IAM](#) en la AWS SDKs Guía de referencia sobre herramientas. En concreto, siga las instrucciones necesarias en No he establecido el acceso a través del Centro de identidades de IAM.

Configuración de SDK para usar IAM Identity Center

La información sobre cómo configurar el SDK para usar el Centro de Identidad de IAM se encuentra en el paso 2 del tema sobre la [autenticación del Centro de Identidad de IAM](#) en la Guía de referencia sobre herramientas AWS SDKs y herramientas. Tras completar esta configuración, el sistema debe contener los siguientes elementos:

- El AWS CLI, que se utiliza para iniciar una sesión en el portal de AWS acceso antes de ejecutar la aplicación.
- El AWS config archivo compartido que contiene un [\[default\]perfil](#) con un conjunto de valores de configuración a los que se puede hacer referencia desde el SDK. Para encontrar la ubicación

de este archivo, consulte [Ubicación de los archivos compartidos](#) en la Guía de referencia de AWS SDKs and Tools. AWS SDK para .NET Utiliza el proveedor del token de inicio de sesión único del perfil para adquirir las credenciales antes de enviar las solicitudes a AWS. El valor `sso_role_name`, que es un rol de IAM conectado a un conjunto de permisos del Centro de identidades de IAM, debería permitir el acceso a los Servicios de AWS utilizados en la aplicación.

El siguiente archivo config de ejemplo muestra la configuración de un perfil predeterminado con el proveedor de token de SSO. La configuración `sso_session` del perfil hace referencia a la sección `sso-session` nombrada. La `sso-session` sección contiene la configuración para iniciar una sesión en el portal de AWS acceso.

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

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

Important

Si lo utilizas AWS IAM Identity Center para la autenticación, tu aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución del SSO funcione:

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

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

Inicie una sesión en el portal de AWS acceso

Antes de ejecutar una aplicación para acceder Servicios de AWS, necesita una sesión activa en el portal de AWS acceso para que el SDK utilice la autenticación del IAM Identity Center a fin de resolver las credenciales. En función de la duración de las sesiones configuradas, el acceso terminará por caducar y SDK detectará un error de autenticación. Para iniciar sesión en el portal de AWS acceso, ejecute el siguiente comando en AWS CLI

```
aws sso login
```

Como tiene una configuración de perfil predeterminada, no necesita llamar al comando con una opción `--profile`. Si la configuración del proveedor de token de SSO utiliza un perfil con nombre, el comando es `aws sso login --profile named-profile`.

Para comprobar si ya tiene una sesión activa, ejecute el siguiente AWS CLI comando.

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

La respuesta a este comando debe indicar la cuenta y el conjunto de permisos del Centro de identidades de IAM configurados en el archivo compartido `config`.

Note

Si ya tiene una sesión activa en el portal de AWS acceso y la ejecuta `aws sso login`, no tendrá que proporcionar credenciales. Es posible que el proceso de inicio de sesión le pida que permita el AWS CLI acceso a sus datos. Como AWS CLI se basa en el SDK para Python, los mensajes de permiso pueden contener variaciones del `botocore` nombre.

Información adicional

- Para obtener más información sobre cómo usar IAM Identity Center y el SSO en un entorno de desarrollo, consulte [Inicio de sesión único](#) en la sección [Autenticación avanzada](#). Esta información incluye métodos alternativos y más avanzados, así como tutoriales que muestran cómo utilizar estos métodos.

- Para obtener más opciones de autenticación para el SDK, como el uso de perfiles y variables de entorno, consulta el capítulo de [configuración](#) de la AWS SDKs Guía de referencia de herramientas.
- Para obtener más información sobre las prácticas recomendadas, consulte [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.
- Para crear AWS credenciales de corta duración, consulte [Credenciales de seguridad temporales](#) en la Guía del usuario de IAM.
- Para obtener más información sobre otros proveedores de credenciales, consulte los proveedores de [credenciales estandarizados en la Guía](#) de referencia de herramientas AWS SDKs y herramientas.

Creación de una aplicación sencilla mediante el AWS SDK para .NET

Esta sección proporciona tutoriales básicos para los desarrolladores que no están familiarizados con AWS SDK para .NET.

Note

Antes de usar estos tutoriales, primero debe haber [instalado su cadena de herramientas](#) y haber [configurado la autenticación de SDK](#).

Para obtener información sobre el desarrollo de software para AWS servicios específicos junto con ejemplos de código, consulte [AWS Servicios de llamadas](#). Para ver otros ejemplos de código, consulte [SDK para .NET ejemplos de código \(v4\)](#).

Temas

- [Aplicación multiplataforma sencilla que utiliza el AWS SDK para .NET](#)
- [Aplicación sencilla basada en Windows que utiliza AWS SDK para .NET](#)
- [Pasos a seguir a continuación](#)

Aplicación multiplataforma sencilla que utiliza el AWS SDK para .NET

En este tutorial se utilizan .NET Core AWS SDK para .NET y .NET Core para el desarrollo multiplataforma. En el tutorial se muestra cómo utilizar SDK para enumerar los [buckets de Amazon S3](#) que posee y, opcionalmente, crear un bucket.

Llevará a cabo este tutorial utilizando herramientas multiplataforma como la interfaz de la línea de comandos (CLI) .NET. Para ver otras formas de configurar el entorno de desarrollo, consulte [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#).

Necesario para el desarrollo multiplataforma de .NET en Windows, Linux o macOS:

- Microsoft [.NET Core SDK](#), versión 2.1, 3.1 o posterior, que incluye la interfaz de línea de comandos (CLI) de .NET (**dotnet**) y el tiempo de ejecución de .NET Core.
- Un editor de código o entorno de desarrollo integrado (IDE) adecuado para el sistema operativo y los requisitos. Normalmente, es uno que proporciona cierto soporte para .NET Core.

Algunos ejemplos son [Microsoft Visual Studio Code \(VS Code\)](#), [JetBrains Rider](#) y [Microsoft Visual Studio](#).

Note

Antes de usar estos tutoriales, primero debe haber [instalado su cadena de herramientas](#) y haber [configurado la autenticación de SDK](#).

Pasos

- [Creación del proyecto](#)
- [Crear el código](#)
- [Ejecución de la aplicación](#)
- [Limpieza](#)

Creación del proyecto

1. Abra un símbolo del sistema o un terminal. Busque o cree una carpeta del sistema operativo en la que pueda crear un proyecto .NET.
2. En esa carpeta, ejecute el siguiente comando para crear el proyecto .NET.

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

3. Vaya a la carpeta S3CreateAndList recién creada y ejecute los siguientes comandos:

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

Los comandos anteriores instalan los NuGet paquetes desde el [administrador de NuGet paquetes](#). Como sabemos exactamente qué NuGet paquetes necesitamos para este tutorial, podemos realizar este paso ahora. También es común que los paquetes requeridos se conozcan durante el desarrollo. Cuando esto sucede, se puede ejecutar un comando similar en ese momento.

Crear el código

1. En la carpeta S3CreateAndList, busque y abra Program.cs en su editor de código.
2. Reemplace el contenido con el siguiente código y guarde el archivo.

```
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;
    }
}
}

```

Ejecución de la aplicación

1. Ejecute el siguiente comando.

```
dotnet run
```

2. Examine la salida para ver el número de buckets de Amazon S3 que posee, si los hay, y sus nombres.

3. Elija un nombre para un nuevo bucket de Amazon S3. Usa «dotnet-quicktour-s3-1-cross-» como base y añádele algo único, como un GUID o tu nombre. Asegúrese de seguir las reglas de nomenclatura de bucket, como se describe en [Reglas de nomenclatura de buckets](#) en la [Guía del usuario de Amazon S3](#).
4. Ejecute el siguiente comando y *amzn-s3-demo-bucket* sustitúyalo por el nombre del depósito que haya elegido.

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

5. Examine la salida para ver el nuevo bucket que se creó.

Limpieza

Mientras realizaba este tutorial, ha creado algunos recursos que puede decidir limpiar en este momento.

- Si no quiere conservar el bucket que la aplicación creó en un paso anterior, elimínelo mediante la consola de Amazon S3 en <https://console.aws.amazon.com/s3/>.
- Si no quiere conservar el proyecto de .NET, elimine la carpeta `S3CreateAndList` del entorno de desarrollo.

Pasos siguientes

Vuelva al [menú de recorrido rápido](#) o vaya directamente al [final de este recorrido rápido](#).

Aplicación sencilla basada en Windows que utiliza AWS SDK para .NET

En este tutorial se utiliza AWS SDK para .NET en Windows con Visual Studio y .NET Core. En el tutorial se muestra cómo utilizar SDK para enumerar los [buckets de Amazon S3](#) que posee y, opcionalmente, crear un bucket.

Llevará a cabo este tutorial en Windows mediante Visual Studio y .NET Core. Para ver otras formas de configurar el entorno de desarrollo, consulte [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#).

Necesario para el desarrollo en Windows con Visual Studio y .NET Core:

- [Microsoft Visual Studio](#)

- Microsoft .NET Core 2.1, 3.1 o posterior

Esto normalmente se incluye de forma predeterminada al instalar una versión reciente de Visual Studio.

Note

Antes de usar estos tutoriales, primero debe haber [instalado su cadena de herramientas](#) y haber [configurado la autenticación de SDK](#).

Pasos

- [Creación del proyecto](#)
- [Crear el código](#)
- [Ejecución de la aplicación](#)
- [Limpieza](#)

Creación del proyecto

1. Abra Visual Studio y cree un nuevo proyecto que utilice la versión C# de la plantilla de la aplicación de consola; es decir, con la descripción: «... para crear una aplicación de línea de comandos que pueda ejecutarse en .NET...». Asigne un nombre al proyecto S3CreateAndList.

Note

No elija la versión .NET Framework de la plantilla de la aplicación de consola o, si la elige, asegúrese de utilizar .NET Framework 4.7.2 o una versión posterior.

2. Con el proyecto recién creado cargado, elija Tools, NuGetPackage Manager y Manage NuGet Packages for Solution.
3. Busque los siguientes NuGet paquetes e instálelos en el proyecto:
AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, y AWSSDK.SSO0IDC

Este proceso instala los NuGet paquetes desde el [administrador de NuGet paquetes](#). Como sabemos exactamente qué NuGet paquetes necesitamos para este tutorial, podemos realizar

este paso ahora. También es común que los paquetes requeridos se conozcan durante el desarrollo. Cuando esto suceda, siga un proceso similar para instalarlos en ese momento.

4. Si tiene la intención de ejecutar la aplicación desde el símbolo del sistema, abra un símbolo del sistema ahora y vaya a la carpeta que contendrá la salida de compilación. Suele ser algo así como `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, pero dependerá del entorno.

Crear el código

1. En el proyecto `S3CreateAndList`, busque y abra `Program.cs` en el IDE.
2. Reemplace el contenido con el siguiente código y guarde el archivo.

```
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. Compilación de la aplicación.

Note

Si usa una versión antigua de Visual Studio, es posible que se produzca un error de compilación similar al siguiente:

“La función ‘async main’ no está disponible en C# 7.0. Utilice la versión 7.1 o superior del lenguaje”.

Si aparece este error, configure el proyecto para que utilice una versión posterior del lenguaje. Por lo general, esto se lleva a cabo en las propiedades del proyecto, en **Compilación > Avanzado**.

Ejecución de la aplicación

1. Ejecute la aplicación sin argumentos de línea de comandos. Haga esto en el símbolo del sistema (si ya tiene abierto uno) o desde el IDE.
2. Examine la salida para ver el número de buckets de Amazon S3 que posee, si los hay, y sus nombres.
3. Elija un nombre para un nuevo bucket de Amazon S3. Usa "dotnet-quicktour-s3-1-winvs-" como base y añádele algo único, como un GUID o tu nombre. Asegúrese de seguir las reglas de nomenclatura de bucket, como se describe en [Reglas de nomenclatura de buckets](#) en la [Guía del usuario de Amazon S3](#).

4. Vuelva a ejecutar la aplicación, esta vez proporcionando el nombre del bucket.

En la línea de comandos, sustituya *amzn-s3-demo-bucket* el siguiente comando por el nombre del bucket que haya elegido.

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

O bien, si está ejecutando la aplicación en el IDE, elija Project, S3 CreateAndList Properties, Debug e introduzca allí el nombre del bucket.

5. Examine la salida para ver el nuevo bucket que se creó.

Limpieza

Mientras realizaba este tutorial, ha creado algunos recursos que puede decidir limpiar en este momento.

- Si no quiere conservar el bucket que la aplicación creó en un paso anterior, elimínelo mediante la consola de Amazon S3 en <https://console.aws.amazon.com/s3/>.
- Si no quiere conservar el proyecto de .NET, elimine la carpeta S3CreateAndList del entorno de desarrollo.

Pasos siguientes

Vuelva al [menú de recorrido rápido](#) o vaya directamente al [final de este recorrido rápido](#).

Pasos a seguir a continuación

Asegúrese de limpiar los recursos sobrantes que haya creado mientras realiza estos tutoriales. Pueden ser AWS recursos o recursos de su entorno de desarrollo, como archivos y carpetas.

Ahora que lo has visitado AWS SDK para .NET, quizá te interese [empezar tu proyecto](#).

Configuración del AWS SDK para .NET

Para usarlo AWS SDK para .NET, debe [configurar su entorno](#) y también configurar una serie de elementos esenciales que su aplicación necesita para acceder a AWS los servicios. Entre ellos se incluyen:

- Especificación de la AWS región
- AWSSDK paquetes o conjuntos

Algunos de los temas de esta sección contienen información sobre cómo configurar estos elementos esenciales.

En otros temas de esta sección y en otras secciones se proporciona información sobre formas más avanzadas de configurar su proyecto.

Temas

- [Inicio de un nuevo proyecto](#)
- [Instale AWSSDK paquetes con NuGet](#)
- [Instale AWSSDK conjuntos sin NuGet](#)
- [Configuración de AWS la región para AWS SDK para .NET](#)
- [Resolución de credencial y perfil](#)
- [Reintentos y tiempos de espera](#)
- [Observabilidad](#)
- [Información adicional acerca de los usuarios y los roles](#)
- [Configuración avanzada para su AWS SDK para .NET proyecto](#)
- [Uso de credenciales heredadas](#)

Inicio de un nuevo proyecto

Existen varias técnicas que puede utilizar para iniciar un nuevo proyecto para acceder a AWS los servicios. Estas son algunas de ellas:

- Si es nuevo en el desarrollo de.NET AWS o, al menos, es nuevo en el AWS SDK para .NET, puede ver ejemplos completos en [Crear una aplicación sencilla](#), que sirve de introducción a SDK.

- Puede iniciar un proyecto básico utilizando la CLI de .NET. Para ver un ejemplo de esto, abra una línea de comandos o un terminal, cree una carpeta o un directorio y acceda a él y, a continuación, introduzca lo siguiente.

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

Se crea un proyecto vacío al que puede añadir código y NuGet paquetes. Para obtener más información, consulte la [Guía de .NET Core](#).

Para ver una lista de plantillas de proyectos, utilice lo siguiente: `dotnet new --list`

- AWS Toolkit for Visual Studio Incluye plantillas de proyectos de C# para una variedad de AWS servicios. Después de [instalar el kit de herramientas](#) en Visual Studio, puede acceder a las plantillas mientras crea un nuevo proyecto.

Para ver esto, vaya a [Trabajar con AWS servicios](#) en la [Guía del AWS Toolkit for Visual Studio usuario](#). Varios de los ejemplos de esa sección sirven para crear nuevos proyectos.

- Si desarrolla con Visual Studio en Windows pero sin él AWS Toolkit for Visual Studio, utilice las técnicas habituales para crear un nuevo proyecto.

Para ver un ejemplo, abra Visual Studio y seleccione Archivo > Nuevo > Proyecto. Busque “.net core” y elija la versión C# de la plantilla Aplicación de consola (.NET Core) o Aplicación WPF (.NET Core). Se crea un proyecto vacío al que puede añadir código y NuGet paquetes.

Puede encontrar algunos ejemplos de cómo trabajar con AWS los servicios en [Ejemplos de código guiados](#).

Important

Si los utilizas AWS IAM Identity Center para la autenticación, tu aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución del SSO funcione:

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

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

Instale AWSSDK paquetes con NuGet

[NuGet](#) es un sistema de administración de paquetes para la plataforma .NET. Con NuGet él, puede instalar los [AWSSDKpaquetes](#), así como varias otras extensiones, en su proyecto. Para obtener información adicional, consulte el repositorio [aws/dotnet](#) en el sitio web. GitHub

NuGet siempre tiene las versiones más recientes de los AWSSDK paquetes, así como las versiones anteriores. NuGet conoce las dependencias entre los paquetes e instala automáticamente todos los paquetes necesarios.

Warning

La lista de NuGet paquetes puede incluir uno que se llame simplemente «AWSSDK» (sin un identificador adjunto). NO instale este NuGet paquete; es un paquete heredado y no debe usarse en nuevos proyectos.

Los paquetes instalados con NuGet almacenan con el proyecto en lugar de en una ubicación central. De este modo podrá instalar versiones de ensamblados específicos de una aplicación determinada sin que se generen problemas de compatibilidad con otras aplicaciones. Para obtener más información al respecto NuGet, consulte la [NuGet documentación](#).

Note

Si no puede o no se le permite descargar e instalar NuGet paquetes por proyecto, puede obtener los AWSSDK ensamblados y almacenarlos localmente (o in situ).

Si este es su caso y aún no ha obtenido los AWSSDK ensamblajes, consulte [Obtención de AWSSDK conjuntos](#). Para obtener información sobre cómo utilizar los ensamblados almacenados localmente, consulte [Instale AWSSDK conjuntos sin NuGet](#).

NuGet Utilizándolo desde la línea de comandos o el terminal

1. Ve a los [AWSSDK paquetes NuGet](#) y determina qué paquetes necesitas en tu proyecto; por ejemplo, [AWSSDK.S3](#).
2. Copie el comando de la CLI de .NET de la página web de ese paquete, como se muestra en el siguiente ejemplo.

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

3. En el directorio del proyecto, ejecute ese comando CLI de .NET. NuGet [también instala cualquier dependencia, como AWSSDK .Core](#).

Note

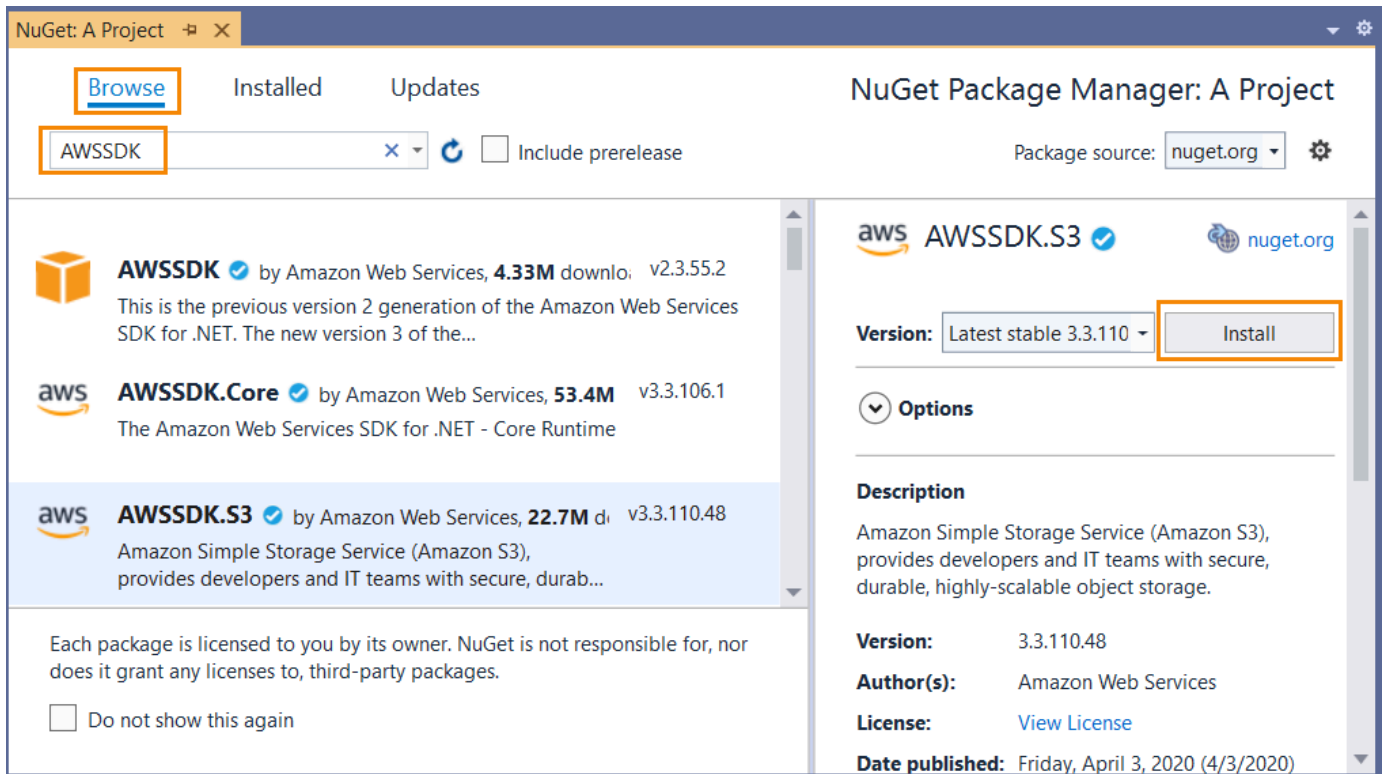
Si solo desea la última versión de un NuGet paquete, puede excluir la información de la versión del comando, como se muestra en el siguiente ejemplo.

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

NuGet Utilizándolo desde el Explorador de soluciones de Visual Studio

1. En el Explorador de soluciones, haga clic con el botón derecho en el proyecto y, a continuación, seleccione Administrar NuGet paquetes en el menú contextual.
2. En el panel izquierdo del NuGet Package Manager, elija Browse. A continuación, puede utilizar el cuadro de búsqueda para buscar el paquete que desea instalar. NuGet [también instala cualquier dependencia, como AWSSDK .Core](#).

La siguiente figura muestra la instalación del paquete .S3. AWSSDK



Uso NuGet desde la consola de Package Manager

En Visual Studio, elija Tools, NuGet Package Manager, Package Manager Console.

Puede instalar los AWSSDK paquetes que desee desde la consola de Package Manager mediante el **Install-Package** comando. Por ejemplo, para instalar [AWSSDK.S3](#), utilice el siguiente comando.

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

NuGet [también instala cualquier dependencia, como .Core. AWSSDK](#)

Si necesita instalar una versión previa de un paquete, utilice la opción `-Version` y especifique la versión del paquete que quiera, como se muestra en el siguiente ejemplo.

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

Para obtener más información sobre los comandos de Package Manager Console, consulte la [PowerShellreferencia](#) en la [NuGetdocumentación](#) de Microsoft.

Instale AWSSDK conjuntos sin NuGet

En este tema se describe cómo puede utilizar los AWSSDK ensamblajes que ha obtenido y almacenado localmente (o in situ), tal y como se describe en [Obtención de AWSSDK conjuntos](#). Este no es el método recomendado para administrar las referencias de SDK, pero en algunos entornos es obligatorio.

Note

El método recomendado para gestionar las referencias del SDK consiste en descargar e instalar solo los NuGet paquetes que necesite cada proyecto. Este método se describe en [Instale AWSSDK paquetes con NuGet](#).

Para instalar AWSSDK ensamblajes

1. Cree una carpeta en el área del proyecto para los AWSSDK ensamblajes necesarios. A modo de ejemplo, podría denominar esta carpeta `AwsAssemblies`
2. Si aún no lo ha hecho, [obtenga los AWSSDK ensamblados](#) y los colocará en alguna carpeta local de descarga o instalación. Copie los archivos DLL de los ensamblados necesarios de esa carpeta de descargas en el proyecto (en la carpeta `AwsAssemblies`, según nuestro ejemplo).

Asegúrese de copiar también las dependencias. Puede encontrar información sobre las dependencias en el [GitHub](#) sitio web.

3. Haga referencia a los ensamblados necesarios de la siguiente manera.

Cross-platform development

1. Abra el archivo `.csproj` del proyecto y agregue un elemento `<ItemGroup>`.
2. En el elemento `<ItemGroup>`, agregue un elemento `<Reference>` con un atributo `Include` para cada ensamblado necesario.

En Amazon S3, por ejemplo, agregaríamos las siguientes líneas al archivo `.csproj` del proyecto.

En Linux y macOS:

```
<ItemGroup>
```

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

En Windows:

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

3. Guarde el archivo .csproj del proyecto.

Windows with Visual Studio and .NET Core

1. En Visual Studio, cargue el proyecto y abra Proyecto > Agregar referencia.
2. Pulse el botón Examinar en la parte inferior del cuadro de diálogo. Navegue hasta la carpeta del proyecto y la subcarpeta en la que copió los archivos DLL necesarios (AwsAssemblies, por ejemplo).
3. Seleccione todos los archivos DLL y, luego, seleccione Agregar y Aceptar.
4. Guarde el proyecto.

Configuración de AWS la región para AWS SDK para .NET

AWS Las regiones le permiten acceder a AWS los servicios que residen físicamente en una región geográfica específica. Esto puede ser útil para evitar redundancias y para que sus datos y aplicaciones se ejecuten cerca del lugar desde donde accederá a ellos usted y sus usuarios.

Para ver la lista actual de todas las regiones y puntos de conexión compatibles con cada AWS servicio, consulte los puntos de conexión [y las cuotas de los servicios](#) en la. Referencia general de AWS Para ver una lista de los puntos de conexión regionales existentes, consulte [Puntos de conexión de servicios de AWS](#). Para ver información detallada sobre las regiones, consulte [Especificar qué regiones de AWS puede usar su cuenta](#).

Puede crear un cliente de AWS servicio que vaya a una región [en particular](#). También puede configurar la aplicación con una región que se utilizará para [todos los clientes AWS de servicio](#). Estos dos casos se explican a continuación.

Creación de un cliente de servicio con una región particular

Puede especificar la región de cualquiera de los clientes de AWS servicio de su aplicación. La configuración de la región de esta manera prevalece sobre cualquier configuración global de ese cliente de servicio concreto.

Región existente

En este ejemplo, se muestra cómo crear una instancia de un [EC2 cliente de Amazon](#) en una región existente. Utiliza campos definidos [RegionEndpoint](#).

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

Nueva región usando la RegionEndpoint clase

En este ejemplo, se muestra cómo construir un nuevo punto final de región mediante [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
}
```

Nueva región mediante la clase de configuración de cliente de servicio

En este ejemplo, se muestra cómo utilizar la ServiceURL propiedad de la clase de configuración del cliente de servicio para especificar la región; en este caso, se utiliza la clase [Amazon EC2 Config](#).

Esta técnica funciona incluso si el punto de conexión de la región no sigue el patrón de punto de conexión de región habitual.

```
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
}
```

Especificación de una región para todos los clientes de servicio

Hay varias formas de especificar una región para todos los clientes de AWS servicio que cree la aplicación. Esta región se usa con los clientes de servicio que no se crean con una región particular.

AWS SDK para .NET Busca un valor de región en el siguiente orden.

Perfiles

Establezca un perfil que la aplicación o SDK haya cargado. Para obtener más información, consulte [Resolución de credencial y perfil](#).

Variables de entorno

Establezca la variable de entorno `AWS_REGION`.

En Linux o macOS:

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

En Windows:

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

Note

Si configura esta variable de entorno para todo el sistema (utilizando `export` o `setx`), afectará a todos los SDKs kits de herramientas, no solo a los AWS SDK para .NET.

AWSConfigs clase

Establecido como un [AWSConfigs.AWSRegion](#) propiedad.


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

Resolución de la región

Si no se utiliza ninguno de los métodos descritos anteriormente para especificar una Región de AWS, AWS SDK para .NET intentará encontrar una región en la que pueda operar el cliente del AWS servicio.

Orden de resolución de la región

1. Archivos de configuración de la aplicación, como `app.config` y `web.config`
2. Variables de entorno (`AWS_REGION` y `AWS_DEFAULT_REGION`)
3. Un perfil con el nombre especificado por un valor en `AWSConfigs.AWSProfileName`
4. Un perfil con el nombre especificado por la variable de entorno `AWS_PROFILE`
5. El perfil `[default]`
6. Metadatos de EC2 instancias de Amazon (si se ejecutan en una EC2 instancia).

Si no se encuentra ninguna región, el SDK lanza una excepción que indica que el cliente del AWS servicio no tiene ninguna región configurada.

Información especial sobre la región de China (Pekín)

Para utilizar servicios en la región de China (Pekín), debe disponer de una cuenta y de credenciales específicas de la región de China (Pekín). Las cuentas y credenciales de otras AWS regiones no funcionarán en la región China (Pekín). Del mismo modo, las cuentas y credenciales de la región de China (Pekín) no funcionarán en otras AWS regiones. Para obtener más información acerca de los puntos de enlace y los protocolos disponibles en la región de China (Pekín), consulte [Puntos finales de la región de Pekín](#).

Información especial sobre nuevos AWS servicios

AWS Los nuevos servicios pueden lanzarse inicialmente en algunas regiones y luego ofrecerse soporte en otras regiones. En estos casos, no es necesario instalar el último SDK para acceder

a las nuevas regiones para ese servicio. Puede especificar las regiones que se han agregado recientemente para cada cliente o de forma global, como se ha indicado anteriormente.

Resolución de credencial y perfil

AWS SDK para .NET Busca las credenciales en un orden determinado y utiliza el primer conjunto disponible para la aplicación actual.

Orden de búsqueda de credenciales

1. Credenciales que se establecen de forma explícita en el cliente AWS de servicio, tal y como se describe en [Acceso a las credenciales y perfiles en una aplicación](#).

Note

Este tema está en la sección [Consideraciones especiales](#) porque no es el método preferido para especificar credenciales.

2. [Sesión AWSCredentials](#) que se crea a partir de las variables de `AWS_SESSION_TOKEN` entorno y `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY`, si las tres variables tienen un valor.
3. [Básicas AWSCredentials](#) que se crean a partir de las variables de `AWS_SECRET_ACCESS_KEY` entorno `AWS_ACCESS_KEY_ID` y, si ambas variables tienen un valor.
4. [AssumeRoleWithWebIdentityCredentials](#) que se crean a partir de las variables de `AWS_ROLE_ARN` entorno `AWS_WEB_IDENTITY_TOKEN_FILE` y, si ambas variables tienen un valor.
5. Un perfil de credenciales con el nombre especificado por un valor en [AWSConfigs](#). [AWSProfile](#)Nombre.
6. Un perfil de credenciales con el nombre especificado por la variable de entorno `AWS_PROFILE`.
7. El perfil de credenciales [default].
8. El [proveedor de credenciales del contenedor](#).
9. Metadatos de EC2 instancias de Amazon.

Si tu aplicación se ejecuta en una EC2 instancia de Amazon, por ejemplo, en un entorno de producción, utiliza un rol de IAM tal y como se describe en [Concesión de acceso mediante un rol de IAM](#). De lo contrario, por ejemplo, en las pruebas preliminares, guarda tus credenciales en un archivo que utilice el formato de archivo de AWS credenciales al que tiene acceso tu aplicación web en el servidor.

Para obtener información adicional sobre las variables de entorno y los perfiles de credenciales, consulte los siguientes temas de la [guía de referencia AWS SDKs y herramientas](#): [variables de entorno](#), [lista de variables de entorno](#) y archivos de [configuración y credenciales compartidos](#).

Resolución de perfiles

Existen dos mecanismos de almacenamiento diferentes para las credenciales, por lo que es importante entender cómo configurarlos AWS SDK para .NET para utilizarlos. El [AWSConfigs](#). [AWSProfiles](#)La propiedad de [ubicación](#) controla la forma en AWS SDK para .NET que busca los perfiles de credenciales.

AWSProfilesLocation	Comportamiento de resolución de perfiles
null (no establecido) o vacío	Busque en SDK Store si la plataforma lo admite y, a continuación, busque el archivo de credenciales de AWS compartido en la ubicación predeterminada . Si el perfil no está en ninguna de esas ubicaciones, busque en <code>~/.aws/config</code> (Linux o macOS) o en <code>%USERPROFILE%\.aws\config</code> (Windows).
La ruta a un archivo en el formato de archivo de AWS credenciales	Busque solo en el archivo especificado el perfil con el nombre indicado.

Uso de credenciales de cuentas de usuario federado

Las aplicaciones que lo utilizan AWS SDK para .NET ([AWSSDK.Core](#) versión 3.1.6.0 y posteriores) pueden usar cuentas de usuario federadas a través de los Servicios de federación de Active Directory (AD FS) para acceder a los AWS servicios mediante el lenguaje de marcado de aserciones de seguridad (SAML).

La compatibilidad del acceso federado significa que los usuarios pueden autenticarse mediante su Active Directory. Las credenciales temporales se conceden al usuario automáticamente. Estas credenciales temporales, que son válidas durante una hora, se utilizan cuando la aplicación invoca los servicios. AWS El SDK gestiona la administración de las credenciales temporales. Para las cuentas de usuario unidas a un dominio, si su aplicación realiza una llamada, pero las credenciales

han caducado, el usuario vuelve a autenticarse automáticamente y se conceden credenciales actualizadas (En el caso de las cuentas que no están unidas a un dominio, se le pide al usuario que introduzca las credenciales antes de volver a autenticarse).

Para utilizar este soporte en su aplicación.NET, primero debe configurar el perfil de rol mediante un PowerShell cmdlet. Para saber cómo, consulte la [documentación de AWS Tools for Windows PowerShell](#).

Después de configurar el perfil del rol, haga referencia al perfil en la aplicación. [Existen varias formas de hacerlo, una de las cuales es mediante el. AWSConfigs AWSProfileAsigne un nombre](#) a la propiedad del mismo modo que lo haría con otros perfiles de credenciales.

[El AWS Security Token Serviceensamblaje \(AWSSDK. SecurityToken\)](#) proporciona el soporte de SAML para obtener AWS credenciales. Para usar credenciales de cuentas de usuario federado, asegúrese de que este ensamblado está disponible para la aplicación.

Especificación de roles o credenciales temporales

En el caso de las aplicaciones que se ejecutan en EC2 instancias de Amazon, la forma más segura de gestionar las credenciales es utilizar funciones de IAM, tal y como se describe en [Concesión de acceso mediante un rol de IAM](#).

Para los escenarios de aplicación en los que el software ejecutable está disponible para los usuarios externos a su organización, recomendamos que diseñe el software para usar credenciales de seguridad temporales. Además de proporcionar un acceso restringido a AWS los recursos, estas credenciales tienen la ventaja de caducar después de un período de tiempo específico. Para obtener más información sobre cómo usar credenciales de seguridad temporales, consulte lo siguiente:

- [Credenciales de seguridad temporales](#)
- [Grupos de identidades de Amazon Cognito](#)

Uso de credenciales de proxy

Si su software se comunica con AWS él a través de un proxy, puede especificar las credenciales del proxy utilizando la `ProxyCredentials` propiedad de la `Config` clase de un servicio.

La clase `Config` de un servicio suele formar parte del espacio de nombres principal del servicio. Los ejemplos incluyen los siguientes: [AmazonCloudDirectoryConfigen Amazon. CloudDirectorynamespace](#) y [AmazonGameLiftConfigen Amazon. GameLift](#) espacio de nombres.

Para [Amazon S3](#), por ejemplo, puede utilizar un código similar al siguiente, donde `SecurelyStoredUserName` y `SecurelyStoredPassword` son el nombre de usuario y la contraseña del proxy especificados en un [NetworkCredential](#) objeto.

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

Note

Las versiones anteriores del SDK usaban `ProxyUsername` y `ProxyPassword`, pero estas propiedades están obsoletas.

Reintentos y tiempos de espera

AWS SDK para .NET Le permite configurar el número de reintentos y los valores de tiempo de espera de las solicitudes HTTP a AWS los servicios. Si los valores predeterminados de los reintentos y los tiempos de espera no son adecuados para su aplicación, puede ajustarlos para sus requisitos específicos, pero es importante comprender cómo afectará hacer esto al comportamiento de su aplicación.

Para determinar qué valores usar para los reintentos y los tiempos de espera, tenga en cuenta lo siguiente:

- ¿Cómo deben responder la aplicación AWS SDK para .NET y la suya cuando la conectividad de la red se deteriora o no se puede acceder a un AWS servicio? ¿Desea que la llamada fracase rápidamente o es adecuado que la llamada se siga reintentando en su nombre?
- ¿Es su aplicación una aplicación o sitio web que se orienta a los usuarios y que debe tener capacidad de respuesta, o se trata de una tarea de procesamiento en segundo plano con una mayor tolerancia de las latencias más elevadas?
- ¿La aplicación está implementada en una red fiable con baja latencia o en una ubicación remota con una conectividad poco fiable?

Reintentos

Descripción general de

AWS SDK para .NET Pueden reintentar las solicitudes que fallan debido a una limitación del lado del servidor o a una interrupción de las conexiones. Hay dos propiedades de las clases de configuración de servicios que se pueden usar para especificar el comportamiento de reintento de un cliente de servicio. [Las clases de configuración de servicios heredan estas propiedades del resumen Amazon.Runtime.ClientConfig](#) [clase de la referencia de la API AWS SDK para .NET](#) :

- `RetryMode` especifica uno de los tres modos de reintento, que se definen en [Amazon.Runtime.RequestRetryMode](#) enumeración.

El valor predeterminado de la aplicación se puede controlar mediante la variable de entorno `AWS_RETRY_MODE` o el ajuste `retry_mode` del archivo de configuración compartido. AWS

- `MaxErrorRetry` especifica el número de reintentos permitidos en el nivel del cliente de servicio; SDK vuelve a intentar la operación el número de veces especificado antes de generar un error y lanzar una excepción.

El valor predeterminado de la aplicación se puede controlar mediante la variable de entorno `AWS_MAX_ATTEMPTS` o la configuración `max_attempts` del archivo de configuración compartido. AWS

[Las descripciones detalladas de estas propiedades se encuentran en el resumen Amazon.Runtime.ClientConfig](#) [clase de la AWS SDK para .NET API Reference](#). Cada valor de `RetryMode` se corresponde de forma predeterminada con un valor concreto de `MaxErrorRetry`, como se muestra en la siguiente tabla.

RetryMode	Correspondiente MaxErrorRetry (Amazon DynamoDB)	Correspondiente MaxErrorRetry (todos los demás)
Legacy	10	4
Standard	10	2
Adaptativo (experimental)	10	2

Comportamiento

Cuando la aplicación se inicia

Cuando la aplicación se inicia, SDK configura los valores predeterminados de `RetryMode` y `MaxErrorRetry`. Estos valores predeterminados se utilizarán cuando cree un cliente de servicio, a menos que especifique otros valores.

- Si las propiedades no están configuradas en el entorno, el valor predeterminado de `RetryMode` se configura como `Heredado` y el valor predeterminado de `MaxErrorRetry` se configura con el valor correspondiente de la tabla anterior.
- Si el modo de reintento está configurado en el entorno, ese valor se utiliza como valor predeterminado de `RetryMode`. El valor predeterminado de `MaxErrorRetry` se configura con el valor correspondiente de la tabla anterior, a menos que el valor de máximo de errores también esté configurado en el entorno (se describe a continuación).
- Si el valor de máximo de errores está configurado en el entorno, ese valor se utiliza como valor predeterminado de `MaxErrorRetry`. Amazon DynamoDB es la excepción a esta regla; el valor de DynamoDB predeterminado de `MaxErrorRetry` es siempre el valor de la tabla anterior.

Mientras la aplicación se ejecuta

Al crear un cliente de servicio, puede usar los valores predeterminados de `RetryMode` y `MaxErrorRetry`, tal y como se ha explicado anteriormente, o bien puede especificar otros valores. Para especificar otros valores, cree e incluya un objeto de configuración de servicio, como [AmazonDynamoDBConfigAmazon](#), `SQSConfig` al crear el cliente de servicio.

Una vez creado el servicio de cliente, estos valores no podrán modificarse.

Consideraciones

Cuando se produce un reintento, aumenta la latencia de la solicitud. Debe configurar sus reintentos según los límites de su aplicación para la latencia de solicitudes total y las tasas de error.

Tiempos de espera

AWS SDK para .NET Esto le permite configurar los tiempos de espera de las solicitudes a nivel del cliente de servicio y por método de llamada. Existen dos mecanismos para configurar los tiempos de espera, que se describen en las secciones siguientes:

- Si utiliza [llamadas asíncronas](#), puede utilizar el parámetro `CancellationToken` del método.
- Si utiliza [llamadas sincrónicas en .NET Framework](#), puede utilizar las `ReadWriteTimeout` propiedades `Timeout` y del resumen `Amazon.Runtime.ClientConfig` clase.

Uso del parámetro **CancellationToken** para los tiempos de espera

AWS SDK para .NET Permite configurar los tiempos de espera de las solicitudes en llamadas asíncronas mediante el parámetro. `CancellationToken` El siguiente fragmento de código muestra un ejemplo. El código genera una excepción `System.Threading.Tasks.TaskCanceledException` si la solicitud no se completa en 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);
}
```

Uso de las propiedades **Timeout** y **ReadWriteTimeout** para los tiempos de espera

Note

La propiedad `Timeout` no afecta a las llamadas asíncronas. Si utiliza llamadas asíncronas, consulte [Uso del parámetro `CancellationToken` para los tiempos de espera](#) en su lugar.

AWS SDK para .NET Esto le permite configurar los valores de tiempo de espera de la solicitud y del tiempo de espera del socket a nivel del read/write cliente del servicio. [Estos valores se](#)

[especifican en las `TimeoutReadWriteTimeout` propiedades del resumen `Amazon.Runtime.ClientConfig`](#) clase. Estos valores se transmiten como `ReadWriteTimeout` propiedades de `Timeout` los [HttpWebRequest](#) objetos creados por el objeto del cliente de AWS servicio. De forma predeterminada, el valor `Timeout` es de 100 segundos y el valor `ReadWriteTimeout` es de 300 segundos.

Cuando su red tiene una elevada latencia o existen condiciones que hacen que se reintente una operación, el uso de valores de tiempo de espera largo y un elevado número de reintentos pueden hacer que algunas operaciones del SDK parezcan no tener capacidad de respuesta.

 Note

La versión de la AWS SDK para .NET que se dirige a la biblioteca de clases portátil (PCL) utiliza la [HttpClient](#) clase en lugar de la `HttpWebRequest` clase y solo admite la propiedad [Timeout](#).

A continuación, se indican las excepciones sobre los valores de tiempo de espera predeterminados. Estos valores se invalidan al establecer de forma explícita los valores de tiempo de espera.

- [Timeout y ReadWriteTimeout se establecen en los valores máximos si el método al que se llama carga una transmisión, como `AmazonS3Client.PutObjectAsync\(\)`, cliente `Amazon S3.UploadPartAsync\(\)`, `AmazonGlacierClient UploadArchiveAsync\(\)`, y así sucesivamente.](#)
- Las versiones de .NET Framework de destino establecidas `Timeout` y `ReadWriteTimeout` con los valores máximos para todos los objetos y [clientes de Amazon S3](#). AWS SDK para .NET [AmazonGlacierClient](#)
- [Las versiones AWS SDK para .NET que se dirigen a la biblioteca de clases portátil \(PCL\) y a .NET Core están configuradas en el valor máximo `Timeout` para todos los objetos y clientes de `AmazonS3Client`. `AmazonGlacierClient`](#)

El siguiente ejemplo muestra cómo especificar el modo de reintento estándar, un máximo de 3 reintentos, un tiempo de espera de 10 segundos y un read/write tiempo de espera de 10 segundos (si corresponde). Al constructor [AmazonS3Client](#) se le asigna un 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  
});
```

Observabilidad

La observabilidad es la medida en que se puede deducir el estado actual de un sistema a partir de los datos que emite. Los datos emitidos se denominan comúnmente “telemetría”.

AWS SDK para .NET Pueden proporcionar dos señales telemétricas comunes, métricas y trazas, así como registros. Puedes configurar un cable [TelemetryProvider](#) para enviar datos de telemetría a un backend de observabilidad (como [AWS X-RayAmazon CloudWatch](#)) y luego actuar en consecuencia.

De forma predeterminada, las señales de telemetría están deshabilitadas en el SDK. En este tema se explica cómo habilitar y configurar la salida de telemetría.

Recursos adicionales

Para obtener más información sobre la habilitación y el uso de la observabilidad, consulte los siguientes recursos:

- [OpenTelemetry](#)
- La entrada del blog [Enhancing Observability](#) in the with. AWS SDK para .NET OpenTelemetry
- La entrada del blog [que anuncia la disponibilidad general de las AWS OpenTelemetry bibliotecas.NET](#).
- [Exportadores para OpenTelemetry](#)
- Para ver ejemplos de observabilidad en el Herramientas de AWS para PowerShell, consulte [Observabilidad](#) en la Guía del [PowerShell usuario de Tools for User](#).

Configuración de un **TelemetryProvider**

Puede configurar un `TelemetryProvider` en su aplicación de forma global para todos los clientes del servicio o para clientes individuales, como se muestra en los siguientes ejemplos. La sección

[the section called “Proveedores de telemetría”](#) contiene información sobre las implementaciones de telemetría, incluida información sobre las implementaciones que se proporcionan con el SDK.

Configuración del proveedor de telemetría global predeterminado

De forma predeterminada, todos los clientes del servicio intentan utilizar el proveedor de telemetría disponible en todo el mundo. De esta forma, puede configurar el proveedor una vez y todos los clientes lo utilizarán. Esto debe hacerse solo una vez, antes de crear cualquier cliente de servicio.

En el siguiente fragmento de código se muestra cómo configurar el proveedor de telemetría global. A continuación, se crea un cliente de servicio de Amazon S3 y se intenta realizar una operación que no funciona. El código añade tanto el rastreo como las métricas a la aplicación. Este código usa los siguientes NuGet paquetes: `OpenTelemetry.Exporter.Console` y `OpenTelemetry.Instrumentation.AWS`

Note

Si lo utilizas AWS IAM Identity Center para la autenticación, asegúrate de añadir también `AWSSDK.SSO` y `AWSSDK.SSOOIDC`.

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

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

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

var s3Client = new AmazonS3Client();
```

```
try
{
    var listBucketsResponse = await s3Client.ListBucketsAsync();
    // Attempt to delete a bucket that doesn't exist.
    var deleteBucketResponse = await s3Client.DeleteBucketAsync("amzn-s3-demo-bucket");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

Console.Read();
```

Configuración de un proveedor de telemetría para un cliente de servicio específico

Puede configurar un cliente de servicio individual con un proveedor de telemetría específico (distinto del global). Para ello, utilice la clase `TelemetryProvider` del objeto `Config` del constructor de un cliente de servicio. Por ejemplo, consulte [AmazonS3Config](#) y busque la propiedad `TelemetryProvider`. Consulte [the section called “Proveedores de telemetría”](#) para obtener información sobre las implementaciones de telemetría personalizadas.

Temas

- [Métricas](#)
- [Proveedores de telemetría](#)

Métricas

En la siguiente tabla se enumeran las métricas de telemetría que emite el SDK. [Configura un proveedor de telemetría](#) para que las métricas sean observables.

¿Qué métricas se emiten?

Nombre de métrica	Unidades	Tipo	Atributos	Descripción
duración de la llamada del cliente	s	Histograma	rpc.service, rpc.método	Duración total de la llamada (incluido los reintentos y el tiempo de envío o recepción de la solicitud y el cuerpo de la respuesta)

Nombre de métrica	Unidades	Tipo	Atributos	Descripción
tiempo de actividad del cliente	s	Histograma	rpc.service	El tiempo transcurrido desde que se creó un cliente
client.call.attempts	{intento}	Monoton Counter	rpc.service, rpc.método	El número de intentos de una operación individual
client.call.errors	{error}	Monoton Counter	rpc.service, rpc.método, exception.type	El número de errores de una operación
client.call.attempt_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en conectarse al servicio, enviar la solicitud y recuperar los encabezados y el código de estado HTTP (incluido el tiempo de espera para ser enviado)
client.call.resolve_endpoint_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en resolver un punto final (el solucionador de puntos finales, no el DNS) de la solicitud
client.call.serialization_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en serializar el cuerpo de un mensaje
client.call.deserialization_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en deserializar el cuerpo de un mensaje
client.call.auth.signing_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en firmar una solicitud
client.call.auth.resolve_identity_duration	s	Histograma	rpc.service, rpc.método	El tiempo que se tarda en adquirir una identidad (por ejemplo, AWS credenciales o un token portador) de un proveedor de identidades

Nombre de métrica	Unidades	Tipo	Atributos	Descripción
client.http.bytes_sent	Mediante	Monoton Counter	dirección del servidor	El número total de bytes enviados por el cliente HTTP
client.http.bytes_received	Mediante	Monoton Counter	dirección del servidor	El número total de bytes recibidos por el cliente HTTP

A continuación se muestran las descripciones de las columnas:

- Nombre de la métrica: el nombre de la métrica emitida.
- Unidades: la unidad de medida de la métrica. Las unidades se indican en la notación [UCUM](#) que distingue entre mayúsculas y minúsculas («c/s»).
- Tipo: el tipo de instrumento utilizado para capturar la métrica.
- Atributos: el conjunto de atributos (dimensiones) emitidos con la métrica.
- Descripción: una descripción de lo que mide la métrica.

Proveedores de telemetría

[El SDK proporciona una implementación OpenTelemetry como proveedor de telemetría, que se describe en la siguiente sección.](#)

Si tiene requisitos de telemetría específicos, ya tiene en mente una solución de telemetría o necesita un control detallado sobre la forma en que se capturan y procesan los datos de telemetría, también puede implementar su propio proveedor de telemetría.

Registre [TelemetryProvider](#) su propia implementación en la clase. El siguiente es un ejemplo sencillo de cómo registrar su propia `TracerProvider` y `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());
```

Temas

- [Configurar el proveedor de telemetría OpenTelemetry basado](#)

Configurar el proveedor de telemetría OpenTelemetry basado

AWS SDK para .NET Incluye una implementación de un proveedor de OpenTelemetry telemetría basado. Para obtener más información sobre cómo configurar este proveedor como proveedor de telemetría global, consulte [Configuración de un TelemetryProvider](#). Para utilizar este proveedor de telemetría, necesita los siguientes recursos en su proyecto:

- El paquete [OpenTelemetry.Instrumentation.aws](#). NuGet
- Un exportador de telemetría, como OTLP o Console. [Para obtener más información, consulte Exportadores en la documentación](#). OpenTelemetry

La OpenTelemetry implementación incluida en el SDK se puede configurar para reducir la cantidad de seguimiento de las solicitudes, las credenciales y la compresión HTTPS. Para ello, defina la opción `SuppressDownstreamInstrumentation` en `true`, de forma similar a la siguiente:

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

Para obtener más información sobre este proveedor, consulta la entrada del blog [Enhancing Observability in the AWS SDK para .NET with](#). OpenTelemetry

Información adicional acerca de los usuarios y los roles

Para desarrollar AWS o ejecutar aplicaciones.NET en ellas AWS, debe tener una combinación de usuarios, conjuntos de permisos y funciones de servicio adecuada para estas tareas.

Los usuarios, conjuntos de permisos y roles de servicio específicos que cree, así como la forma en que los utilice, dependerán de los requisitos de las aplicaciones. A continuación, se muestra información adicional sobre por qué se podrían usar y cómo crearlos.

Usuarios y conjuntos de permisos

Aunque es posible utilizar una cuenta de usuario de IAM con credenciales de larga duración para acceder a los servicios de AWS , esta práctica ya no es recomendable y se debe evitar. Incluso durante el desarrollo, se recomienda crear usuarios y conjuntos de permisos AWS IAM Identity Center y utilizar las credenciales temporales proporcionadas por una fuente de identidad.

Para el desarrollo, puede usar el usuario que ha creado o que le han proporcionado en [Autenticarse con AWS](#). Si tiene Consola de administración de AWS los permisos adecuados, también puede crear diferentes conjuntos de permisos con los privilegios mínimos para ese usuario o crear nuevos usuarios específicamente para proyectos de desarrollo, proporcionando conjuntos de permisos con los privilegios mínimos. El curso de acción que elija, si corresponde, depende de las circunstancias.

Para obtener más información sobre estos usuarios y conjuntos de permisos y sobre cómo crearlos, consulte [Autenticación y acceso](#) en la AWS SDKs Guía de referencia sobre herramientas y [Introducción](#) en la Guía del AWS IAM Identity Center usuario.

Roles de servicio

Puede configurar un rol de AWS servicio para acceder a AWS los servicios en nombre de los usuarios. Este tipo de acceso es adecuado si varias personas van a ejecutar tu aplicación de forma remota; por ejemplo, en una EC2 instancia de Amazon que hayas creado para este fin.

El proceso de creación de un rol de servicio varía en función de la situación, pero básicamente es el siguiente.

1. Inicie sesión en la consola de IAM Consola de administración de AWS y ábrala en <https://console.aws.amazon.com/iam/>.
2. Seleccione Roles y, a continuación, Crear rol.

3. Elija el AWS servicio, busque y seleccione EC2(por ejemplo) y, a continuación, elija el caso de EC2uso (por ejemplo).
4. Elija Siguiente: permisos y seleccione las [políticas adecuadas](#) para los AWS servicios que utilizará la aplicación.

**Warning**

NO elija la AdministratorAccesspolítica porque esa política permite permisos de lectura y escritura para casi todo el contenido de su cuenta.

5. Seleccione Siguiente: Etiquetas e introduzca las etiquetas que desee.

Puede encontrar información sobre las etiquetas en [Controlar el acceso mediante etiquetas de AWS recursos](#) en la [Guía del usuario de IAM](#).

6. Seleccione Siguiente: Revisar y proporcione un nombre de rol y una descripción del rol. A continuación, elija Crear rol.

Puede encontrar información general sobre los roles de IAM en [Identities de IAM \(usuarios, grupos de usuarios y roles\)](#) en la [Guía del usuario de IAM](#). Encuentre información detallada sobre los roles en el tema [Roles de IAM](#) de esa guía.

Información adicional sobre los roles

- Use [roles de IAM para tareas](#) para tareas de Amazon Elastic Container Service (Amazon ECS).
- Utilice [funciones de IAM](#) para las aplicaciones que se ejecutan en EC2 instancias de Amazon.

Configuración avanzada para su AWS SDK para .NET proyecto

Los temas de esta sección contienen información sobre más tareas y métodos de configuración que podrían interesarle.

Temas

- [Uso de AWSSDK.Extensions.NETCore.Setup y la interfaz IConfiguration](#)
- [Configuración de otros parámetros de la aplicación](#)
- [Referencia de archivos de configuración para AWS SDK para .NET](#)

Uso de AWSSDK.Extensions.NETCore.Setup y la interfaz IConfiguration

(Este tema se titulaba anteriormente «Configuración AWS SDK para .NET con .NET Core»)

Uno de los mayores cambios en .NET Core es la eliminación de ConfigurationManager y de los archivos app.config y web.config estándar que se usaban con aplicaciones de .NET Framework y ASP.NET.

La configuración en .NET Core se basa en pares clave-valor establecidos por proveedores de configuración. Los proveedores de configuración leen los datos de configuración en pares clave-valor desde diversos orígenes de configuración, incluidos argumentos de línea de comandos, archivos de directorio, variables de entorno y archivos de configuración.

Note

Para obtener más información, consulte [Configuración en ASP.NET Core](#).

Para facilitar su uso AWS SDK para .NET con .NET Core, puede utilizar las [AWSSDK.Extensions.NETCore.Paquete de configuración](#). NuGet Al igual que muchas bibliotecas de .NET Core, añade métodos de extensión a la IConfiguration interfaz para facilitar la AWS configuración.

El código fuente de este paquete está GitHub en <https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup>.

Uso de AWSSDK.Extensions.NETCore.Configuración

Suponga que crea una aplicación ASP.NET Core Model-View-Controller (MVC), que puede realizarse con la plantilla de aplicación web ASP.NET Core en Visual Studio o ejecutándola dotnet new mvc ... en la CLI de .NET Core. Al crear una aplicación de este tipo, el constructor de Startup.cs administra la configuración leyendo varios orígenes de entrada de proveedores de configuración, como appsettings.json.

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

Para usar el `Configuration` objeto para obtener las `AWS` opciones, primero agregue el paquete. `AWSSDK.Extensions.NETCore.Setup` NuGet. Luego, agregue sus opciones al archivo de configuración como se describe a continuación.

Fíjese en que uno de los archivos agregados al proyecto es `appsettings.Development.json`. Corresponde a un `EnvironmentName` establecido en `Development`. Durante el desarrollo, pondremos la configuración en este archivo, que solo se lee durante las pruebas locales. Al implementar una instancia de Amazon EC2 `EnvironmentName` configurada en `Producción`, este archivo se ignora y se recurre a las credenciales AWS SDK para .NET de IAM y a la región configuradas para la instancia de Amazon EC2.

Los siguientes ajustes de configuración muestran ejemplos de los valores que se pueden agregar en el archivo `appsettings.Development.json` del proyecto para proporcionar la configuración de AWS.

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

Para obtener acceso a un ajuste en un archivo CSHTML utilice la directiva `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 acceder a las AWS opciones definidas en el archivo desde el código, llame al método de `GetAWSOptions` extensión agregado a `IConfiguration`

Para crear un cliente de servicio a partir de estas opciones, llame a `CreateServiceClient`. En el siguiente ejemplo se muestra cómo crear un cliente de servicio de Amazon S3 (Asegúrese de añadir el [AWSSDKpaquete.S3](#) NuGet a su proyecto).

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

También puede crear varios clientes de servicio con ajustes incompatibles utilizando varias entradas en el archivo `appsettings.Development.json`, tal y como se muestra en los siguientes ejemplos, donde la configuración de `service1` incluye la región `us-west-2` y la configuración de `service2` incluye la URL del punto de conexión especial.

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

A continuación, puede obtener las opciones para un servicio específico mediante la entrada del archivo JSON. Por ejemplo, utilice lo siguiente para obtener la configuración de `service1`.

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

Valores permitidos en el archivo `appsettings`

Los siguientes valores de configuración de la aplicación se pueden establecer en el archivo `appsettings.Development.json`. Los nombres de los campos deben escribirse con las mayúsculas y minúsculas mostradas aquí. Para obtener más información sobre esta configuración, consulte la clase [AWS.Runtime.ClientConfig](#).

- Region
- Profile
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp

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

Inserción de dependencias de ASP.NET Core

Las `AWSSDK.Extensions.NETCore.Setup` NuGet El paquete `.Setup` también se integra con un nuevo sistema de inyección de dependencias en ASP.NET Core. El método `ConfigureServices` de la clase `Startup` de la aplicación es donde se agregan los servicios MVC. Si la aplicación usa Entity Framework, también es donde se inicializa.

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

Note

Hay disponible información general sobre la inserción de dependencias en .NET Core en el [sitio de documentación de .NET Core](#).

El `AWSSDK.Extensions.NETCore.Setup` NuGet paquete agrega nuevos métodos de extensión `IServiceCollection` que puede usar para agregar AWS servicios a la inyección de dependencias. El siguiente código muestra cómo añadir las AWS opciones de lectura

IConfiguration para añadir Amazon S3 y DynamoDB a la lista de servicios. (Asegúrese de añadir los DBv2 NuGet paquetes [AWSSDK.S3](#) y [AWSSDK.Dynamo](#) a su proyecto).

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

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

Ahora, si sus controladores MVC usan IAmazonS3 o IAmazonDynamoDB como parámetros en sus constructores, el sistema de inserción de dependencias pasa esos servicios.

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

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

    ...
}
```

Configuración de otros parámetros de la aplicación

Note

La información de este tema es específica de proyectos basados en .NET Framework. Los archivos App.config y Web.config no están presentes de forma predeterminada en proyectos basados en .NET Core.

Abrir para ver el contenido de .NET Framework

Hay varios parámetros de la aplicación que se pueden configurar:

- [AWSLogging](#)
- [AWSLogMetrics](#)
- [AWSRegion](#)
- [AWSResponseLogging](#)
- [AWS.DynamoDBContext.TableNamePrefix](#)
- [AWS.S3.UseSignatureVersion4](#)
- [AWSEndpointDefinition](#)
- [AWS Terminales generados por el servicio](#)

Estos parámetros se pueden configurar en el archivo `App.config` o `Web.config` de la aplicación. Aunque también puedes configurarlos con la AWS SDK para .NET API, te recomendamos que utilices el archivo de la aplicación. `.config` Aquí se describen ambos enfoques.

Para obtener más información sobre cómo usar el elemento `<aws>` tal como se describe más adelante en este tema, consulte [Referencia de archivos de configuración de AWS SDK para .NET](#).

AWSLogging

Configura la manera en que el SDK debe registrar los eventos, si es que debe hacerlo. Por ejemplo, el enfoque recomendado es utilizar el elemento `<logging>`, que es un elemento secundario del elemento `<aws>`:

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

Otra opción:

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

Los valores posibles son:

None

Desactivación del registro de eventos. Este es el valor predeterminado.

log4net

Registro mediante log4net.

SystemDiagnostics

Registro mediante la clase `System.Diagnostics`.

Puede definir diferentes valores para el atributo `logTo`, separado por comas. El siguiente ejemplo define los registros `log4net` y `System.Diagnostics` en el archivo `.config`:

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

Otra opción:

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

Como alternativa, mediante la AWS SDK para .NET API, combine los valores de la [LoggingOptions](#) enumeración y establezca la propiedad [AWSConfigs.Logging](#):

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

Los cambios en esta configuración solo se aplican a las instancias de AWS cliente nuevas.

AWSLogMétricas

Especifica si el SDK debería o no registrar métricas de desempeño. Para definir la configuración de registro de las métricas en el archivo `.config`, defina el valor del atributo `logMetrics` en el elemento `<logging>`, que es un elemento secundario del elemento `<aws>`:

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

De forma alternativa, defina la clave `AWSLogMetrics` en la sección `<appSettings>`:

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

Como alternativa, para configurar el registro de métricas con la AWS SDK para .NET API, configure [AWSConfigs.LogMetrics](#) propiedad:

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


Esta opción configura la propiedad `LogMetrics` de forma predeterminada para todos los clientes y configuraciones. Los cambios en esta configuración solo se aplican a las instancias de AWS clientes nuevas.

AWSRegion

Configura la AWS región predeterminada para los clientes que no han especificado una región de forma explícita. Para definir la región en el archivo `.config`, le recomendamos especificar el valor del atributo `region` en el elemento `aws`:

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

De forma alternativa, defina la clave `AWSRegion` en la sección `<appSettings>`:

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

[Como alternativa, para configurar la región con la AWS SDK para .NET API, configure. `AWSConfigs.AWSRegion` propiedad:](#)

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

Para obtener más información sobre la creación de un AWS cliente para una región específica, consulte [Selección de AWS regiones](#). Los cambios en esta configuración solo se aplican a las instancias de AWS clientes nuevas.

AWSResponseRegistro

Se configura cuando el SDK debería registrar respuestas del servicio. Los valores posibles son:

Never

No registrar nunca respuestas del servicio. Es el valor predeterminado.

Always

Registrar siempre respuestas del servicio.

OnError

Registrar solo respuestas del servicio cuando se produzcan errores.

Para definir la configuración del registro de servicios en el archivo `.config`, le recomendamos que defina el valor del atributo `logResponses` en el elemento `<logging>`, que es un elemento secundario del elemento `<aws>`:

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

También puede configurar la clave de `AWSResponseLogging` en la `<appSettings>` sección:

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

Como alternativa, para configurar el registro del servicio con la AWS SDK para .NET API, configure [AWSConfigs.ResponseLogging](#) propiedad en uno de los valores de la [ResponseLoggingOption](#) enumeración:

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

Los cambios en esta configuración surtirán efecto de inmediato.

AWS.DynamoDBContext.TableNamePrefix

Configura el `TableNamePrefix` predeterminado. Se utilizará `DynamoDBContext` si no se ha configurado manualmente.

Para definir el prefijo del nombre de tabla en el archivo `.config`, le recomendamos definir el valor del atributo `tableNamePrefix` en el elemento `<dynamoDBContext>`, que es un elemento secundario del elemento `<dynamoDB>`, que a su vez es un elemento secundario del elemento `<aws>`:

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

De forma alternativa, defina la clave `AWS.DynamoDBContext.TableNamePrefix` en la sección `<appSettings>`:

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

Como alternativa, para establecer el prefijo del nombre de la tabla con la AWS SDK para .NET API, establezca la propiedad [AWSConfigsDBContextTableNamePrefix.Dynamo](#):

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

Los cambios en esta configuración solo serán efectivos en las instancias `DynamoDBContextConfig` y `DynamoDBContext` creadas recientemente.

AWS.S3.UseSignatureVersion4

Configura si el cliente de Amazon S3 debería utilizar o no la versión 4 de Signature con solicitudes.

Para establecer la versión 4 de Signature para Amazon S3 en el archivo `.config`, el método recomendado es establecer el valor del atributo `useSignatureVersion4` del elemento `<s3>`, que es un elemento secundario del elemento `<aws>`:

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

De forma alternativa, establezca la clave `AWS.S3.UseSignatureVersion4` en `true` en la sección `<appSettings>`:

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

Como alternativa, para configurar la firma en la versión 4 de la firma con la AWS SDK para .NET API, establezca la [AWSConfigspropiedad.S3 UseSignatureVersion](#) 4 en: `true`

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

De forma predeterminada, esta configuración es `false`, pero la versión 4 de Signature puede utilizarse de forma predeterminada en algunos casos o en algunas regiones. Si la configuración es `true`, se utilizará la versión 4 de Signature para todas las solicitudes. Los cambios en esta configuración solo surten efecto en las instancias de cliente de Amazon S3 nuevas.

AWSEndpointDefinición

Configura si el SDK debería utilizar un archivo de configuración personalizado que define las regiones y los puntos de enlace.

Para definir el archivo de definición del punto de enlace en el archivo `.config`, le recomendamos configurar el valor del atributo `endpointDefinition` en el elemento `<aws>`.

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

Como alternativa, puede configurar la clave de AWSEndpointdefinición en la <appSettings> sección:

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

Como alternativa, para configurar el archivo de definición del punto final con la AWS SDK para .NET API, configure el [AWSConfigs.EndpointDefinition](#) propiedad:

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

Si no se proporciona ningún nombre de archivo, no se utilizará el archivo de configuración personalizado. Los cambios en esta configuración solo se aplican a las instancias de AWS clientes nuevas. El endpoints.json archivo está disponible en <https://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json>.

AWS Puntos finales generados por el servicio

Algunos AWS servicios generan sus propios puntos de enlace en lugar de consumir un punto final regional. Los clientes de estos servicios consumen una URL de servicio específica de dicho servicio y sus recursos. Dos ejemplos de estos servicios son Amazon CloudSearch y AWS IoT. Los siguientes ejemplos muestran cómo puede obtener los puntos de enlace para dichos servicios.

Ejemplo de Amazon CloudSearch Endpoints

El CloudSearch cliente de Amazon se utiliza para acceder al servicio de CloudSearch configuración de Amazon. Utiliza el servicio de CloudSearch configuración de Amazon para crear, configurar y gestionar los dominios de búsqueda. Para crear un dominio de búsqueda, cree un [CreateDomainRequest](#) objeto y proporcione la DomainName propiedad. Cree un [AmazonCloudSearchClient](#) objeto mediante el objeto de solicitud. Llame al método [CreateDomain](#). El [CreateDomainResponse](#) objeto devuelto por la llamada contiene una DomainStatus propiedad que tiene los SearchService puntos finales DocService y los extremos. Crea un [AmazonCloudSearchDomainConfig](#) objeto y úsalo para inicializar cualquier SearchService instancia DocService de la [AmazonCloudSearchDomainClient](#) clase.

```
// 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 Ejemplo de puntos finales

Para obtener el punto final AWS IoT, cree un [AmazonIoTClient](#) objeto y llame al [DescribeEndPoint](#) método. El [DescribeEndPointResponse](#) objeto devuelto contiene el `EndpointAddress`. Cree un [AmazonIotDataConfig](#) objeto, establezca la `ServiceURL` propiedad y utilice el objeto para crear una instancia de la [AmazonIotDataClient](#) clase.

```
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");
}
```

Referencia de archivos de configuración para AWS SDK para .NET

Note

La información de este tema es específica de proyectos basados en .NET Framework. Los archivos `App.config` y `Web.config` no están presentes de forma predeterminada en proyectos basados en .NET Core.

Abrir para ver el contenido de .NET Framework

Puede usar un `Web.config` archivo `App.config` o un proyecto de .NET para especificar la AWS configuración, como AWS las credenciales, las opciones de registro, los puntos finales del AWS servicio y AWS las regiones, así como algunas configuraciones de los AWS servicios, como Amazon DynamoDB EC2, Amazon y Amazon S3. En la siguiente información se describe cómo dar formato correctamente a un archivo `App.config` o `Web.config` para especificar estos tipos de configuración.

Note

Si bien puede seguir utilizando el `<appSettings>` elemento de un `Web.config` archivo `App.config` o para especificar la AWS configuración, le recomendamos que utilice los `<aws>` elementos `<configSections>` y tal como se describe más adelante en este tema. Para obtener más información sobre el `<appSettings>` elemento, consulte los ejemplos de `<appSettings>` elementos en [Configuración de AWS SDK para .NET la aplicación](#).

Note

Si bien puede seguir utilizando las siguientes propiedades de `AWSConfigs` clase en un archivo de código para especificar la AWS configuración, las siguientes propiedades están en desuso y es posible que no se admitan en futuras versiones:

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

En general, se recomienda que, en lugar de utilizar las propiedades de `AWSConfigs` clase en un archivo de código para especificar la AWS configuración, utilice los `<aws>` elementos `<configSections>` y de un `Web.config` archivo `App.config` o para especificar la AWS configuración, tal y como se describe más adelante en este tema. Para obtener más

información sobre las propiedades anteriores, consulte los ejemplos AWSConfigs de código en [Configuración de AWS SDK para .NET la aplicación](#).

Temas

- [Declaración de una sección AWS de configuración](#)
- [Elementos permitidos](#)
- [Referencia de elementos](#)

Declaración de una sección AWS de configuración

Los AWS ajustes de un Web.config archivo App.config o se especifican desde dentro del <aws> elemento. Antes de que pueda empezar a usar el elemento <aws>, debe crear un elemento <section> (que es un elemento secundario del elemento <configSections>) y establecer su atributo name en aws y su atributo type en Amazon.AWSSection, AWSSDK.Core, como se muestra en el siguiente ejemplo:

```
<?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>
```

El editor de Visual Studio no proporciona la finalización automática del código (IntelliSense) para el <aws> elemento o sus elementos secundarios.

Para ayudarle a crear una versión con formato correcto del elemento <aws>, llame al método Amazon.AWSConfigs.GenerateConfigTemplate. De este modo se devuelve una versión canónica del elemento <aws> como una cadena bien escrita, que puede adaptar a sus necesidades. En las siguientes secciones se describen los atributos y elementos secundarios del elemento <aws>.

Elementos permitidos

A continuación se muestra una lista de las relaciones lógicas entre los elementos permitidos en una sección de configuración de AWS . Puede generar la versión más reciente de esta lista llamando al método `Amazon.AWSConfigs.GenerateConfigTemplate`, que devuelve una versión canónica del elemento `<aws>` como una cadena que puede adaptar a sus necesidades.

```
<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>
```

Referencia de elementos

La siguiente es una lista de los elementos que están permitidos en una sección AWS de configuración. Para cada elemento, se listan sus atributos y elementos principales y secundarios permitidos.

Temas

- [alias](#)
- [aws](#)
- [dynamoDB](#)
- [dinamo DBContext](#)
- [ec2](#)
- [registrar](#)
- [map](#)
- [propiedad](#)
- [proxy](#)
- [s3](#)

alias

El elemento `<alias>` representa un solo elemento de una colección de uno o varios mapeos a la tabla y desde la tabla que especifica una tabla distinta de una configurada para un tipo. Este elemento se asigna a una instancia de la clase `Amazon.Util.TableAlias` a partir de la propiedad `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` en el AWS SDK para .NET. La reasignación se lleva a cabo antes de la aplicación de un prefijo de nombre de tabla.

Este elemento puede incluir los siguientes atributos:

fromTable

La parte desde la tabla del mapeo a la tabla y desde la tabla. Este atributo se asigna a la propiedad `Amazon.Util.TableAlias.FromTable` en el AWS SDK para .NET.

toTable

La parte a la tabla de la asignación a la tabla y desde la tabla. Este atributo se asigna a la propiedad `Amazon.Util.TableAlias.ToTable` en el AWS SDK para .NET.

El elemento principal del elemento `<alias>` es el elemento `<tableAliases>`.

El elemento `<alias>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<alias>` en uso:

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

aws

El `<aws>` elemento representa el elemento situado en la parte superior de una sección AWS de ajustes. Este elemento puede incluir los siguientes atributos:

endpointDefinition

La ruta absoluta a un archivo de configuración personalizado que define AWS las regiones y los puntos finales que se van a utilizar. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.EndpointDefinition` en el AWS SDK para .NET.

profileName

El nombre del perfil de AWS las credenciales almacenadas que se utilizarán para realizar llamadas de servicio. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.AWSProfileName` en el AWS SDK para .NET.

profilesLocation

La ruta absoluta a la ubicación del archivo de credenciales compartido con otros AWS SDKs. De forma predeterminada, el archivo de credenciales se almacena en el directorio

.aws del directorio de inicio del usuario actual. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.AWSProfilesLocation` en el AWS SDK para .NET.

region

El identificador de AWS región predeterminado para los clientes que no han especificado una región de forma explícita. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.AWSRegion` en el AWS SDK para .NET.

El elemento `<aws>` no tiene elemento principal.

El elemento `<aws>` puede incluir los siguientes elementos secundarios:

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

A continuación, mostramos un ejemplo del elemento `<aws>` en uso:

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

dynamoDB

El elemento `<dynamoDB>` representa una colección de ajustes de Amazon DynamoDB. Este elemento puede incluir el atributo `conversionSchema`, que representa la versión que se va a usar para la conversión entre objetos de .NET y DynamoDB. Los valores permitidos son V1 y V2. Este atributo se asigna a la clase `Amazon.DynamoDBv2.DynamoDBEntryConversion` en el AWS SDK para .NET. Para obtener más información, consulte [Serie DynamoDB: esquemas de conversión](#).

El elemento principal del elemento `<dynamoDB>` es el elemento `<aws>`.

El elemento <dynamoDB> puede incluir el elemento secundario <dynamoDBContext>.

A continuación, mostramos un ejemplo del elemento <dynamoDB> en uso:

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

dinamo DBContext

El elemento <dynamoDBContext> representa una colección de ajustes específicos del contexto de Amazon DynamoDB. Este elemento puede incluir el tableNamePrefix atributo, que representa el prefijo de nombre de tabla predeterminado que utilizará el contexto de DynamoDB si no se configura manualmente. Este atributo se asigna a la propiedad Amazon.Util.DynamoDBContextConfig.TableNamePrefix a partir de la propiedad Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix del AWS SDK para .NET. Para obtener más información, consulte la sección sobre las [mejoras realizadas en el SDK de DynamoDB](#).

El elemento principal del elemento <dynamoDBContext> es el elemento <dynamoDB>.

El elemento <dynamoDBContext> puede incluir los siguientes elementos secundarios:

- <alias> (una o varias instancias)
- <map> (una o varias instancias)

A continuación, mostramos un ejemplo del elemento <dynamoDBContext> en uso:

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

ec2

El <ec2> elemento representa una colección de EC2 ajustes de Amazon. Este elemento puede incluir el atributo useSignatureVersion4, que especifica si la firma de la versión 4 de la firma se utilizará para todas las solicitudes (verdadero) o si la firma de la versión 4 de la firma no se utilizará para todas las solicitudes (falsa, la opción predeterminada). Este atributo se asigna a

la propiedad `Amazon.Util.EC2Config.UseSignatureVersion4` a partir de la propiedad `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` del AWS SDK para .NET.

El elemento principal del elemento `<ec2>` es el elemento.

El elemento `<ec2>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<ec2>` en uso:

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

registrar

El elemento `<logging>` representa una colección de ajustes para el registro de respuestas y el registro de métricas de desempeño. Este elemento puede incluir los siguientes atributos:

logMetrics

Si las métricas de desempeño se van a registrar para todos los clientes y configuraciones (`true`); de lo contrario, `false`. Este atributo se asigna a la propiedad `Amazon.Util.LoggingConfig.LogMetrics` a partir de la propiedad `Amazon.AWSConfigs.LoggingConfig.LogMetrics` del AWS SDK para .NET.

logMetricsCustomFormatter

Tipo de datos y nombre del ensamblado de un formateador personalizado para registrar métricas. Este atributo se asigna a la propiedad `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` a partir de la propiedad `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` del AWS SDK para .NET.

logMetricsFormat

Formato en que se presentan las métricas de registro (se asigna a la propiedad `Amazon.Util.LoggingConfig.LogMetricsFormat` a partir de la propiedad `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` del AWS SDK para .NET).

Entre los valores permitidos se incluyen:

JSON

Utilizar el formato JSON.

Standard

Utilizar el formato predeterminado.

logResponses

Cuándo se deben registrar respuestas del servicio (se asigna a la propiedad `Amazon.Util.LoggingConfig.LogResponses` a partir de la propiedad `Amazon.AWSConfigs.LoggingConfig.LogResponses` en el AWS SDK para .NET).

Entre los valores permitidos se incluyen:

Always

Registrar siempre respuestas del servicio.

Never

No registrar nunca respuestas del servicio.

OnError

Registrar solo respuestas del servicio cuando haya errores.

logTo

Dónde iniciar sesión (se asigna a la `LogTo` propiedad desde la `Amazon.AWSConfigs.LoggingConfig.LogTo` propiedad que se encuentra en AWS SDK para .NET).

Los valores permitidos incluyen una o varias de estas opciones:

Log4Net

Registrarse en log4net.

None

Deshabilitar el registro.

SystemDiagnostics

Registrarse en `System.Diagnostics`.

El elemento principal del elemento `<logging>` es el elemento `<aws>`.

El elemento `<logging>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<logging>` en uso:

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

map

El `<map>` elemento representa un único elemento de una colección de type-to-table asignaciones de tipos .NET a tablas de DynamoDB (se asigna a una instancia de la `TypeMapping` clase desde la propiedad `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` AWS SDK para .NET). Para obtener más información, consulte la sección sobre las [mejoras realizadas en el SDK de DynamoDB](#).

Este elemento puede incluir los siguientes atributos:

targetTable

Tabla de DynamoDB a la que se aplica el mapeo. Este atributo se asigna a la propiedad `Amazon.Util.TypeMapping.TargetTable` en el AWS SDK para .NET.

type

Tipo y nombre del ensamblado al que se aplica el mapeo. Este atributo se asigna a la propiedad `Amazon.Util.TypeMapping.Type` en el AWS SDK para .NET.

El elemento principal del elemento `<map>` es el elemento `<dynamoDBContext>`.

El elemento `<map>` puede incluir una o varias instancias del elemento secundario `<property>`.

A continuación, mostramos un ejemplo del elemento `<map>` en uso:

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



```
</map>
```

propiedad

El elemento `<property>` representa una propiedad DynamoDB. (Este elemento se asigna a una instancia de `Amazon.Util.PropertyConfig` [clase del `AddProperty` método de AWS SDK para .NET](#)) [Para obtener más información, consulte `Mejoras en el SDK de DynamoDB y los atributos de DynamoDB`.](#)

Este elemento puede incluir los siguientes atributos:

attribute

Nombre de un atributo para la propiedad, como el nombre de una clave de rango. Este atributo se asigna a la propiedad `Amazon.Util.PropertyConfig.Attribute` en el AWS SDK para .NET.

converter

Tipo de convertidor que se debe usar para esta propiedad. Este atributo se asigna a la propiedad `Amazon.Util.PropertyConfig.Converter` en el AWS SDK para .NET.

ignore

Si debe hacerse caso omiso de la propiedad asociada (`true`); de lo contrario, `false`. Este atributo se asigna a la propiedad `Amazon.Util.PropertyConfig.Ignore` en el AWS SDK para .NET.

name

El nombre de la propiedad. Este atributo se asigna a la propiedad `Amazon.Util.PropertyConfig.Name` en el AWS SDK para .NET.

version

Si esta propiedad debe almacenar el número de versión del elemento (`true`); de lo contrario, `false`. Este atributo se asigna a la propiedad `Amazon.Util.PropertyConfig.Version` en el AWS SDK para .NET.

El elemento principal del elemento `<property>` es el elemento `<map>`.

El elemento `<property>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<property>` en uso:

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

proxy

El elemento `<proxy>` representa los ajustes para configurar un proxy que AWS SDK para .NET usará. Este elemento puede incluir los siguientes atributos:

host

Nombre de host o dirección IP del servidor proxy. Estos atributos se asignan a la propiedad `Amazon.Util.ProxyConfig.Host` a partir de la propiedad `Amazon.AWSConfigs.ProxyConfig.Host` del AWS SDK para .NET.

contraseña

Contraseña para autenticarse con el servidor proxy. Estos atributos se asignan a la propiedad `Amazon.Util.ProxyConfig.Password` a partir de la propiedad `Amazon.AWSConfigs.ProxyConfig.Password` del AWS SDK para .NET.

puerto

Número de puerto del proxy. Estos atributos se asignan a la propiedad `Amazon.Util.ProxyConfig.Port` a partir de la propiedad `Amazon.AWSConfigs.ProxyConfig.Port` del AWS SDK para .NET.

nombre de usuario

Nombre de usuario para autenticarse con el servidor proxy. Estos atributos se asignan a la propiedad `Amazon.Util.ProxyConfig.Username` a partir de la propiedad `Amazon.AWSConfigs.ProxyConfig.Username` del AWS SDK para .NET.

El elemento principal del elemento `<proxy>` es el elemento `<aws>`.

El elemento `<proxy>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<proxy>` en uso:

```
<proxy
  host="192.0.2.0"
```

```
port="1234"  
username="My-Username-Here"  
password="My-Password-Here" />
```

s3

El elemento `<s3>` representa una colección de ajustes de Amazon S3. Este elemento puede incluir el atributo `useSignatureVersion4`, que especifica si la firma de la versión 4 de la firma se utilizará para todas las solicitudes (`true`) o si la firma de la versión 4 de la firma no se utilizará para todas las solicitudes (`false`, la opción predeterminada). Este atributo se asigna a la propiedad `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` en el AWS SDK para .NET.

El elemento principal del elemento `<s3>` es el elemento `<aws>`.

El elemento `<s3>` no contiene ningún elemento secundario.

A continuación, mostramos un ejemplo del elemento `<s3>` en uso:

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

Uso de credenciales heredadas

En los temas de esta sección se proporciona información sobre el uso de credenciales a corto o largo plazo sin usar AWS IAM Identity Center.

Warning

Para evitar riesgos de seguridad, no utilice a los usuarios de IAM para la autenticación cuando desarrolle software especialmente diseñado o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como [AWS IAM Identity Center](#).

Note

La información de este tema se refiere a las circunstancias en las que necesita obtener y administrar credenciales a corto o largo plazo de forma manual. Para obtener información adicional sobre las credenciales a corto y largo plazo, consulte [Otras formas de autenticación](#) en la Guía de referencia de AWS SDKs and Tools.

Para conocer las mejores prácticas de seguridad AWS IAM Identity Center, utilícelas como se describe en [Autenticarse con AWS](#).

Advertencias y directrices importantes para las credenciales

Advertencias para las credenciales

- NO utilices las credenciales raíz de tu cuenta para acceder a AWS los recursos. Estas credenciales proporcionan acceso ilimitado a la cuenta y son difíciles de revocar.
- NO incluya claves de acceso literales ni información sobre credenciales en sus archivos de aplicación. Si lo hace, puede crear un riesgo de exposición accidental de sus credenciales si, por ejemplo, carga el proyecto en un repositorio público.
- NO incluya archivos que contengan credenciales en el área del proyecto.
- Tenga en cuenta que todas las credenciales almacenadas en el AWS credentials archivo compartido se almacenan en texto sin formato.

Guía adicional para administrar las credenciales de forma segura

Para obtener información general sobre cómo administrar las credenciales de forma segura, consulte AWS las credenciales de [AWS seguridad](#) en la Guía del usuario de IAM [Referencia general de AWS](#) y las [prácticas recomendadas y los casos de uso de seguridad](#) en la Guía del [usuario de IAM](#). Además de esas conversaciones, tenga en cuenta lo siguiente:

- Cree usuarios adicionales, como los usuarios del Centro de identidades de IAM y utilice sus credenciales en lugar de las credenciales de usuario raíz de AWS . Las credenciales de otros usuarios se pueden revocar si es necesario o son de naturaleza temporal. Además, puede aplicar una política a cada usuario para acceder solo a determinados recursos y acciones y, por lo tanto, adoptar una política de permisos con privilegios mínimos.
- Use [roles de IAM para tareas](#) para tareas de Amazon Elastic Container Service (Amazon ECS).
- Use [roles de IAM](#) para aplicaciones que se ejecutan en instancias de Amazon EC2.
- Use [credenciales temporales](#) o variables de entorno de aplicaciones disponibles para los usuarios ajenos a su organización.

Temas

- [Uso del archivo de AWS credenciales compartido](#)
- [Uso de SDK Store \(solo Windows\)](#)

Uso del archivo de AWS credenciales compartido

(Asegúrese de revisar las [advertencias y las directrices de credenciales importantes](#)).

Una forma de proporcionar credenciales para las aplicaciones consiste en crear perfiles en el archivo de credenciales de AWS compartido y, a continuación, almacenar las credenciales en esos perfiles. Este archivo puede ser utilizado por el otro AWS SDKs. También lo pueden usar los [AWS CLI](#) AWS kits de herramientas de [Visual Studio](#) y [VS Code](#). [AWS Tools for Windows PowerShell](#) [JetBrains](#)

Warning

Para evitar riesgos de seguridad, no utilice a los usuarios de IAM para la autenticación cuando desarrolle software especialmente diseñado o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como [AWS IAM Identity Center](#).

Note

La información de este tema se refiere a las circunstancias en las que necesita obtener y administrar credenciales a corto o largo plazo de forma manual. Para obtener información adicional sobre las credenciales a corto y largo plazo, consulte [Otras formas de autenticación](#) en la Guía de referencia de herramientas AWS SDKs y herramientas.

Para conocer las mejores prácticas de seguridad AWS IAM Identity Center, utilícelas como se describe en [Autenticarse con AWS](#).

Información general

De forma predeterminada, el archivo de AWS credenciales compartidas se encuentra en el .aws directorio de su directorio principal y tiene un nombre `credentials`, es decir, `~/.aws/credentials` (Linux o macOS) o `%USERPROFILE%\.aws\credentials` (Windows). Para obtener información sobre ubicaciones alternativas, consulte [Ubicación de los archivos compartidos](#) en la

[Guía de referencia de herramientas AWS SDKs y herramientas](#). Consulte también [Acceso a las credenciales y perfiles en una aplicación](#).

El archivo de AWS credenciales compartidas es un archivo de texto sin formato y tiene un formato determinado. Para obtener información sobre el formato de los archivos de AWS credenciales, consulte [Formato del archivo de credenciales](#) en la Guía de referencia de AWS SDKs and Tools.

Puede administrar los perfiles del archivo de AWS credenciales compartido de varias maneras.

- Utilice cualquier editor de texto para crear y actualizar el archivo de AWS credenciales compartidas.
- Utilice [Amazon.Runtime.CredentialManagement](#) espacio de nombres de la AWS SDK para .NET API, como se muestra más adelante en este tema.
- [Utilice comandos y procedimientos para los Herramientas de AWS para PowerShellAWS kits de herramientas de Visual Studio y VS JetBrainsCode.](#)
- Con comandos de [AWS CLI](#); por ejemplo, `aws configure set aws_access_key_id` y `aws configure set aws_secret_access_key`

Ejemplos de administración de perfiles

En las siguientes secciones se muestran ejemplos de perfiles en el archivo de AWS credenciales compartidas. Algunos de los ejemplos muestran el resultado, que se puede obtener mediante cualquiera de los métodos de administración de credenciales mencionados anteriormente. En otros ejemplos se muestra cómo utilizar un método en particular.

El perfil predeterminado.

El archivo de AWS credenciales compartidas casi siempre tendrá un perfil denominado default. Aquí es donde AWS SDK para .NET busca las credenciales si no hay otros perfiles definidos.

El perfil [default] suele parecerse a lo siguiente.

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

Creación de un perfil mediante programación

En este ejemplo, se muestra cómo crear un perfil y guardarlo en el archivo de AWS credenciales compartidas mediante programación. [Utiliza las siguientes clases de](#)

[Amazon.Runtime.CredentialManagement](#) espacio de nombres: [CredentialProfileOptions](#), y. [CredentialProfileSharedCredentialsFile](#)

```
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

Por lo general, un código como este no debería estar en la aplicación. Si lo incluye en su aplicación, tome las debidas precauciones para garantizar que las claves de texto sin formato no puedan verse en el código, en la red o incluso en la memoria del ordenador.

Este es el perfil que se ha creado en este ejemplo.

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

Actualización de un perfil existente mediante programación

En este ejemplo se muestra cómo actualizar mediante programación el perfil que creado anteriormente. [Utiliza las siguientes clases de Amazon.Runtime.CredentialManagement](#) espacio de

nombres: y. [CredentialProfileSharedCredentialsFile](#) También usa la [RegionEndpoint](#) clase del espacio de nombres [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 continuación se muestra el perfil actualizado.

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

Note

También puede configurar la AWS región en otras ubicaciones y mediante otros métodos. Para obtener más información, consulte [Configuración de AWS la región para AWS SDK para .NET](#).

Uso de SDK Store (solo Windows)

(Asegúrese de revisar las [advertencias y directrices importantes](#)).

En Windows, la tienda de SDK es otro lugar para crear perfiles y almacenar credenciales cifradas para AWS SDK para .NET la aplicación. Está en %USERPROFILE%\AppData\Local\AWSToolkit

\RegisteredAccounts.json. Puede usar SDK Store durante el desarrollo como alternativa al [archivo de credenciales de AWS compartido](#).

Warning

Para evitar riesgos de seguridad, no utilice a los usuarios de IAM para la autenticación cuando desarrolle software especialmente diseñado o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como [AWS IAM Identity Center](#).

Note

La información de este tema se refiere a las circunstancias en las que necesita obtener y administrar credenciales a corto o largo plazo de forma manual. Para obtener información adicional sobre las credenciales a corto y largo plazo, consulta [Otras formas de autenticación](#) en la Guía de referencia de herramientas AWS SDKs y herramientas.

Para conocer las mejores prácticas de seguridad AWS IAM Identity Center, utilícelas como se describe en [Autenticarse con AWS](#).

Información general

SDK Store reporta las siguientes ventajas:

- Las credenciales almacenadas en SDK Store se cifran, y SDK Store reside en el directorio de inicio del usuario. Esto limita el riesgo de exposición accidental de sus credenciales.
- SDK Store también proporciona credenciales a [AWS Tools for Windows PowerShell](#) y a [AWS Toolkit for Visual Studio](#).

Los perfiles de SDK Store son específicos de un usuario determinado en un host concreto. No puede copiarlos en otros hosts ni en otros usuarios. Esto quiere decir que los perfiles de SDK Store que estén en su equipo de desarrollo no se pueden reutilizar en otros hosts o equipos de desarrollador. Esto significa también que no se pueden usar perfiles de SDK Store en aplicaciones de producción.

Los perfiles de SDK Store se pueden administrar de varias maneras:

- Con la interfaz de usuario gráfica (GUI) del [AWS Toolkit for Visual Studio](#)

- Utilice [Amazon.Runtime.CredentialManagement](#) espacio de nombres de la AWS SDK para .NET API, como se muestra más adelante en este tema.
- Con comandos de [AWS Tools for Windows PowerShell](#); por ejemplo, `Set-AWSCredential` y `Remove-AWSCredentialProfile`

Ejemplos de administración de perfiles

En los siguientes ejemplos se muestra cómo crear y actualizar un perfil mediante programación en SDK Store.

Creación de un perfil mediante programación

En este ejemplo se muestra cómo crear un perfil y guardarlo en SDK Store mediante programación.

[Utiliza las siguientes clases de Amazon.Runtime.CredentialManagement namespace: CredentialProfileOptions, CredentialProfile Net File. SDKCredentials](#)

```
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

Por lo general, un código como este no debería estar en la aplicación. Si lo está, tome las debidas precauciones para garantizar que las claves de texto sin formato no puedan verse en el código, en la red o incluso en la memoria del ordenador.

Este es el perfil que se ha creado en este ejemplo.

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

Actualización de un perfil existente mediante programación

En este ejemplo se muestra cómo actualizar mediante programación el perfil que creado anteriormente. [Utiliza las siguientes clases de Amazon.Runtime.CredentialManagement namespace: CredentialProfile Net File. SDKCredentials](#) También usa la [RegionEndpoint](#) clase del espacio de nombres [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 continuación se muestra el perfil actualizado.

```
"[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

También puede configurar la AWS región en otras ubicaciones y mediante otros métodos. Para obtener más información, consulte [Configuración de AWS la región para AWS SDK para .NET](#).

Uso del AWS SDK para .NET

En esta sección se proporciona información sobre las características AWS SDK para .NET que quizás deba tener en cuenta al crear sus aplicaciones.

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#).

Para obtener información sobre el desarrollo de software para AWS servicios específicos junto con ejemplos de código, consulte [AWS Servicios de llamadas](#). Para ver otros ejemplos de código, consulte [SDK para .NET ejemplos de código \(v4\)](#).

Temas

- [Programar de forma asíncrona mediante el AWS SDK para .NET](#)
- [Uso de resultados paginados en AWS SDK para .NET](#)
- [Support para HTTP 2 en el AWS SDK para .NET](#)
- [Herramientas adicionales](#)

Programar de forma asíncrona mediante el AWS SDK para .NET

AWS SDK para .NET Utiliza el patrón asíncrono basado en tareas (TAP) para su implementación asíncrona. Para obtener más información sobre el TAP, consulte [Patrón asincrónico basado en tareas \(TAP\)](#) en docs.microsoft.com.

En este tema se ofrece una descripción general de cómo utilizar el TAP en las llamadas a los clientes del servicio. AWS

Los métodos asíncronos de la AWS SDK para .NET API son operaciones basadas en la Task clase o la clase. Task<TResult> [Consulte docs.microsoft.com para obtener información sobre estas clases: clase Tarea, Task< > clase. TResult](#)

Cuando estos métodos de API se invocan en el código, se deben llamar dentro de una función que esté declarada con la palabra clave async, como se muestra en el siguiente ejemplo.

```
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;
}
```

Como se muestra en el fragmento de código anterior, el ámbito preferido de la declaración de `async` es la función `Main`. Establecer este ámbito `async` garantiza que todas las llamadas a clientes de servicio de AWS van a ser asincrónicas obligatoriamente. Si `Main` no se puede declarar como asincrónico por algún motivo, se puede usar la palabra clave `async` en otras funciones distintas de `Main` y, a continuación, llamar a los métodos de la API desde esas funciones, como se muestra en el siguiente ejemplo.

```
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;
}
```

Fíjese en la sintaxis de `Task<>` especial que se necesita en `Main` al usar este patrón. Además, para obtener los datos, debe usar el miembro **Result** de la respuesta.

Puede ver ejemplos completos de llamadas asincrónicas a clientes de AWS servicio en la sección (y) y en. [Crear una aplicación sencilla](#) [Aplicación multiplataforma sencilla](#) [Aplicación sencilla basada en Windows](#) [Ejemplos de código guiados](#)

Uso de resultados paginados en AWS SDK para .NET

Algunos AWS servicios recopilan y almacenan una gran cantidad de datos, que puede recuperar mediante las llamadas a la API del AWS SDK para .NET. Si la cantidad de datos que se quiere recuperar es demasiado grande para una sola llamada a la API, los resultados se pueden dividir en fragmentos más fáciles de administrar mediante el uso de la paginación.

Para poder realizar la paginación, los objetos de solicitud y de respuesta de muchos clientes de servicio de SDK proporcionan un token de continuación (que suele denominarse `NextToken`). Algunos de estos clientes de servicio también proporcionan paginadores.

Los paginadores permiten evitar la sobrecarga que supone el uso del token de continuación, que puede implicar bucles, variables de estado, múltiples llamadas a la API, etc. Cuando se utiliza un paginador, se pueden recuperar datos de un servicio de AWS con una sola línea de código, mediante la declaración de un bucle `foreach`. Si se necesitan varias llamadas a la API para recuperar los datos, el paginador se encarga de ello por ti.

¿Dónde puedo encontrar paginadores?

No todos los servicios ofrecen paginadores. Una forma de determinar si un servicio proporciona un paginador para una API concreta consiste en consultar la definición de una clase de cliente de servicio en la [Referencia de API de AWS SDK para .NET](#).

Por ejemplo, si examina la definición de la [AmazonCloudWatchLogsClient](#) clase, verá una `Paginate` propiedad. Esta es la propiedad que proporciona un paginador para Amazon CloudWatch Logs.

¿Qué me proporcionan los paginadores?

Los paginadores contienen propiedades que permiten ver respuestas completas. También suelen contener una o más propiedades que permiten acceder a las partes más interesantes de las respuestas, que denominaremos resultados clave.

Por ejemplo, en lo `AmazonCloudWatchLogsClient` mencionado anteriormente, el `Paginator` objeto contiene una `Responses` propiedad con el [DescribeLogGroupsResponse](#) objeto completo de

la llamada a la API. Esta propiedad `Responses` contiene, entre otras cosas, una colección de los grupos de registros.

El objeto `Paginator` contiene también un resultado clave denominado `LogGroups`. Esta propiedad contiene solo la parte de grupos de registros de la respuesta. Tener este resultado clave permite reducir y simplificar el código en muchas ocasiones.

Paginación sincrónica frente a paginación asincrónica

Los paginadores proporcionan mecanismos de paginación sincrónicos y asincrónicos. La paginación sincrónica está disponible en los proyectos de .NET Framework 4.7.2 (o versiones posteriores). La paginación asíncrona está disponible en los proyectos de .NET Core (.NET Core 3.1, .NET 5, etc.).

Dado que se recomienda el uso de operaciones asincrónicas y .NET Core, en el siguiente ejemplo se muestra la paginación asincrónica. La información sobre cómo realizar las mismas tareas mediante la paginación sincrónica y .NET Framework 4.7.2 (o una versión posterior) se muestra después del ejemplo de [Consideraciones adicionales sobre los paginadores](#)

Ejemplo

En el siguiente ejemplo, se muestra cómo utilizar el AWS SDK para .NET para mostrar una lista de grupos de registros. Para mostrar las diferencias, en el ejemplo se muestra cómo hacerlo con y sin paginadores. Antes de echar un vistazo al código completo, que se muestra más adelante, analice los siguientes fragmentos de código.

Obtener grupos de CloudWatch registros sin 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));
```

Obtener grupos de CloudWatch registros mediante 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);
}
```

Los resultados de estos dos fragmentos de código son exactamente iguales, por lo que se ve claramente la ventaja de utilizar paginadores.

Note

Antes de intentar compilar y ejecutar el código completo, asegúrese de haber [configurado el entorno y el proyecto](#).

[Es posible que también necesite el Microsoft.Bcl. AsyncInterfaces](#) NuGet paquete porque los paginadores asíncronos utilizan la interfaz. `IAsyncEnumerable`

Código completo

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.CloudWatch](#)

Elementos de programación:

- [Espacio de nombres Amazon. CloudWatch](#)

Clase [AmazonCloudWatchLogsClient](#)

- [Espacio de nombres Amazon. CloudWatchLogs](#).Modelo

Clase [DescribeLogGroupsRequest](#)

Clase [DescribeLogGroupsResponse](#)

Clase [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}");
        }
        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($"Metadata: {response.ResponseMetadata}");
        Console.WriteLine("Log groups:");
        foreach(LogGroup logGroup in response.LogGroups)
        {
            Console.WriteLine($"\\t{logGroup.LogGroupName}");
        }
    }
}
```

Consideraciones adicionales sobre los paginadores

- Los paginadores no se pueden usar más de una vez

Si necesita los resultados de un AWS paginador en particular en varias ubicaciones de su código, no debe usar un objeto paginador más de una vez. En su lugar, cree un paginador nuevo cada vez que lo necesite. Este concepto se muestra en el código de ejemplo anterior del método `DisplayLogGroupsWithPaginators`.

- Paginación sincrónica

La paginación sincrónica está disponible para proyectos de .NET Framework 4.7.2 (o versiones posteriores).

Para verlo, cree un proyecto de .NET Framework 4.7.2 (o posterior) y cópiele el código anterior. A continuación, simplemente elimine la palabra clave `await` de las dos llamadas al paginador `foreach`, como se muestra en el siguiente ejemplo.

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

Compile y ejecute el proyecto para obtener los mismos resultados que vimos en la paginación asíncrona.

Support para HTTP 2 en el AWS SDK para .NET

Algunos AWS servicios y operaciones requieren HTTP 2. Por ejemplo, la transmisión bidireccional en Amazon Transcribe Streaming no es posible a través de HTTP 1.1 y, por lo tanto, requiere HTTP 2 en su lugar. La versión 4 del soporte AWS SDK para .NET agregado para HTTP 2 le permite usar estas operaciones en sus aplicaciones. En el caso de una operación HTTP 2 bidireccional, el comportamiento del SDK al recibir transmisiones es similar al de HTTP 1.1. Es decir, cuando las aplicaciones que usan el SDK envían eventos al servicio, la solicitud tiene un publicador asignado por el desarrollador.

Para ver este comportamiento en acción, considere el siguiente ejemplo de Amazon Transcribe Streaming. Utiliza [Amazon. TranscribeStreaming](#) y [Amazon. TranscribeStreaming.Espacios de nombres de modelos](#).

En este ejemplo, el desarrollador define la `StartStreamTranscriptionRequest.AudioStreamPublisher` propiedad con una función de devolución de llamada, que es `.NET. Func`. El SDK usa el `Func` valor definido `AudioStreamPublisher` para extraer eventos del código del usuario y transmitirlos al usuario. El SDK llama al `Func` hasta que devuelve un valor nulo.

El código demuestra cómo se puede transmitir el audio de un archivo a Amazon Transcribe Streaming para su procesamiento.

```
using Amazon;
using Amazon.TranscribeStreaming;
using Amazon.TranscribeStreaming.Model;

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

var startRequest = new StartStreamTranscriptionRequest
{
    LanguageCode = LanguageCode.EnUS,
    MediaEncoding = MediaEncoding.Flac,
    MediaSampleRateHertz = 44100,
    NumberOfChannels = 2,
    EnableChannelIdentification = true
};

Stream fileStream = File.OpenRead("hello-world.flac");
```

```

var buffer = new byte[1024 * 10];
startRequest.AudioStreamPublisher += async () =>
{
    var bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);

    if (bytesRead == 0)
        return null;

    var audioEvent = new AudioEvent
    {
        AudioChunk = new MemoryStream(buffer, 0, bytesRead)
    };

    return audioEvent;
};

using var response = await client.StartStreamTranscriptionAsync(startRequest);
Console.WriteLine(response.HttpStatusCode);

response.TranscriptResultStream.ExceptionReceived +=
    TranscriptResultStream_ExceptionReceived;
response.TranscriptResultStream.TranscriptEventReceived +=
    TranscriptResultStream_TranscriptEventReceived;

void TranscriptResultStream_ExceptionReceived(object? sender,
    Amazon.Runtime.EventStreams.EventStreamExceptionReceivedArgs<TranscribeStreamingEventStreamException> e)
{
    Console.WriteLine(e.EventStreamException.Message);
    cancelSource.Cancel();
}

void TranscriptResultStream_TranscriptEventReceived(object? sender,
    Amazon.Runtime.EventStreams.EventStreamEventReceivedArgs<TranscriptEvent> e)
{
    foreach (var result in e.EventStreamEvent.Transcript.Results)
    {
        if (!string.Equals("ch_0", result.ChannelId,
            StringComparison.OrdinalIgnoreCase))
            continue;

        var text = result.Alternatives[0].Transcript;
        if (!string.IsNullOrEmpty(text))
        {
            Console.WriteLine(text);
        }
    }
}

```

```
    }  
  }  
}  
  
_ = response.TranscriptResultStream.StartProcessingAsync();  
  
try  
{  
    await Task.Delay(10000, cancellationToken);  
}  
catch (TaskCanceledException) { }
```

Warning

Algunas operaciones HTTP 2 bidireccionales, como el `InvokeModelWithBidirectionalStreamAsync` método de Amazon Bedrock, Amazon BedrockRuntime espacio de nombres, no devuelve una respuesta de la invocación inicial en el cliente del servicio hasta que se hayan publicado algunos eventos. Este comportamiento podría provocar el bloqueo de la aplicación. Para evitar esta situación, separe el código de la aplicación que proporciona los eventos al publicador y ejecútelo en un subproceso diferente del subproceso que invoca las operaciones en el cliente del servicio.

Consideraciones adicionales

- AWS SDK para .NET La compatibilidad con HTTP 2 solo está disponible en las versiones orientadas a .NET 8 y versiones posteriores. No está disponible en las versiones orientadas a .NET Framework.
- Para obtener información más detallada, consulte [PR 3730](#) en el [aws-sdk-net](#) GitHub repositorio.

Herramientas adicionales

Estas son algunas herramientas adicionales que puede utilizar para facilitar la tarea de desarrollo, implementación y mantenimiento de sus aplicaciones .NET.

AWS Herramienta de implementación

Una vez que haya desarrollado su aplicación .NET Core nativa de la nube en una máquina de desarrollo, puede usar la herramienta de AWS implementación de la CLI de .NET para implementar la aplicación más fácilmente en AWS.

Para obtener más información, consulte [Implemente aplicaciones en AWS](#).

AWS Marco de procesamiento de mensajes para .NET

Si utiliza servicios como Amazon SQS, Amazon SNS o EventBridge Amazon, es posible que pueda aprovechar el marco de procesamiento AWS de mensajes para .NET. Para obtener más información, consulte [AWS Marco de procesamiento de mensajes para .NET](#).

Integraciones con .NET Aspire

Puede aprovechar las integraciones con .NET Aspire para mejorar el ciclo interno de desarrollo. Para obtener más información, consulte [Integración AWS con .NET Aspire en AWS SDK para .NET](#).

Autenticación y autorización avanzadas con AWS SDK para .NET

Los temas de esta sección proporcionan información sobre las técnicas avanzadas de autenticación y autorización en AWS SDK para .NET la aplicación.

Temas

- [Inicio de sesión único con AWS SDK para .NET](#)

Inicio de sesión único con AWS SDK para .NET

AWS IAM Identity Center es un servicio de inicio de sesión único (SSO) basado en la nube que facilita la administración centralizada del acceso SSO a todas sus aplicaciones y a las de la nube. Cuentas de AWS Para obtener más información, consulte la [Guía del usuario de IAM Identity Center](#).

Si no está familiarizado con la forma en que un SDK interactúa con IAM Identity Center, consulte la siguiente información.

Patrón de interacción general

A un nivel alto, SDKs interactúe con el Centro de Identidad de IAM de una manera similar a la siguiente:

1. IAM Identity Center se configura (normalmente, a través de la [consola de IAM Identity Center](#)) y se invita a un usuario de SSO a participar.
2. El AWS config archivo compartido del ordenador del usuario se actualiza con la información del SSO.
3. El usuario inicia sesión a través del Centro de identidad de IAM y recibe credenciales de corta duración para los permisos AWS Identity and Access Management (de IAM) que se le han configurado. Este inicio de sesión se puede iniciar mediante una herramienta ajena al SDK, como la AWS CLI, o mediante programación mediante una aplicación.NET.
4. El usuario se pone a trabajar. Cuando ejecute otras aplicaciones que usen SSO, no necesitará volver a iniciar sesión para abrirlas.

En lo que queda de este tema se proporciona información de referencia para configurar y usar AWS IAM Identity Center. Muestra información complementaria y más avanzada que la configuración

básica de SSO suministrada en [Autenticarse con AWS](#). Si es la primera vez que utilizas el SSO AWS, te recomendamos que consultes primero ese tema para obtener información fundamental y, después, consultes los siguientes tutoriales para ver el SSO en acción:

- [Tutorial: solo aplicación .NET](#)
- [Tutorial: AWS CLI y aplicación.NET](#)

Este tema contiene las siguientes secciones:

- [Requisitos previos](#)
- [Configuración de un perfil de SSO](#)
- [Generación y uso de tokens de SSO](#)
- [Recursos adicionales](#)
- [Tutoriales](#)

Requisitos previos

Antes de utilizar el Centro de identidades de IAM, debe realizar determinadas tareas, como elegir una fuente de identidad y configurar las aplicaciones y aplicaciones pertinentes Cuentas de AWS . Para obtener más información, consulte los siguientes temas:

- Para obtener información general sobre estas tareas, consulte [Introducción](#) en la Guía del usuario de IAM Identity Center.
- Para ver ejemplos concretos de tareas, consulte la lista de tutoriales al final de este tema. No obstante, procure revisar la información de este tema antes de probar con los tutoriales.

Configuración de un perfil de SSO

Una vez [configurado](#) el Centro de Identidad de IAM en la ubicación correspondiente Cuenta de AWS, se debe añadir un perfil con nombre para el inicio de sesión único al archivo compartido del usuario. AWS config Este perfil se utiliza para conectarse al [portal de acceso a AWS](#), donde se obtienen unas credenciales de duración limitada de los permisos de IAM configurados para el usuario.

El archivo config suele denominarse %USERPROFILE%\aws\config en Windows y ~/.aws/config en Linux y macOS. Puede usar el editor de texto de su elección para agregar un nuevo perfil de SSO. También puede utilizar el comando `aws configure sso`. Para obtener más información

sobre este comando, consulte [Configurar la AWS CLI para usar IAM Identity Center](#) en la Guía del usuario de AWS Command Line Interface .

El nuevo perfil es similar al siguiente:

```
[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
```

A continuación se definen los ajustes del nuevo perfil. Los dos primeros ajustes definen el portal de AWS acceso. Los otros dos ajustes son unos ajustes que, en conjunto, definen los permisos que se han configurado para un usuario. Estos cuatro ajustes son obligatorios.

sso_start_url

URL que apunta al [portal de acceso de AWS](#) de la organización. Para encontrar este valor, abra la [consola de IAM Identity Center](#), seleccione Configuración y busque URL de portal.

sso_region

El Región de AWS que contiene el host del portal de acceso. Es la región que se seleccionó al habilitar IAM Identity Center. Puede ser una región distinta de las que se usan en otras tareas.

Para obtener una lista completa de ellos Regiones de AWS y sus códigos, consulte los [puntos finales regionales](#) en el Referencia general de Amazon Web Services.

sso_account_id

El ID de un elemento Cuenta de AWS que se agregó a través del AWS Organizations servicio. Para ver la lista de cuentas disponibles, vaya a la [consola del IAM Identity Center](#) y abra la página Cuentas de AWS. El ID de cuenta que seleccione en este ajuste se corresponderá con el valor que tenga pensado asignar al ajuste sso_role_name, que se muestra a continuación.

sso_role_name

Nombre de un conjunto de permisos de IAM Identity Center. Este conjunto de permisos define los permisos que se otorgan a un usuario a través de IAM Identity Center.

El siguiente procedimiento es una de las formas para encontrar el valor de este ajuste.

1. Vaya a la [consola de IAM Identity Center](#) y abra la página Cuentas de AWS.

2. Seleccione una cuenta para ver sus detalles. La cuenta que seleccione será la que contenga el usuario o grupo de SSO al que quiera conceder permisos de SSO.
3. Consulte la lista de usuarios y grupos asignados a la cuenta y busque el usuario o grupo de su interés. El conjunto de permisos que especifique en el ajuste `sso_role_name` es uno de los conjuntos asociados a este usuario o grupo.

Al asignar un valor a este ajuste, utilice el nombre del conjunto de permisos, no el nombre de recurso de Amazon (ARN).

Los conjuntos de permisos tienen asociadas políticas de IAM y políticas de permisos personalizados. Para obtener más información, consulte [Conjuntos de permisos](#) en la Guía del usuario de IAM Identity Center.

Generación y uso de tokens de SSO

Para usar el SSO, el usuario primero debe generar un token temporal y, a continuación, usarlo para acceder a AWS las aplicaciones y los recursos correspondientes. En el caso de las aplicaciones .NET, puede utilizar los siguientes métodos para generar y utilizar los siguientes tokens temporales:

- Crear aplicaciones .NET que generen primero un token (si es necesario) y, luego, usar el token
- Genere un token con el token AWS CLI y, a continuación, utilícelo en las aplicaciones.NET.

Estos métodos se describen en las siguientes secciones y se explican en los [tutoriales](#).

Important

La aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución del SSO funcione:

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

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

Solo aplicación .NET

En esta sección se muestra cómo crear una aplicación .NET que genera un token de SSO temporal, si es necesario, y luego lo usa. Para ver un tutorial completo de este proceso, consulte [Tutorial de SSO utilizando únicamente aplicaciones .NET](#).

Generación y uso de un token de SSO mediante programación

Además de usar el AWS CLI, también puedes generar un token de SSO mediante programación.

Para ello, la aplicación crea un objeto [AWSCredentials](#) del perfil de SSO, que carga las credenciales temporales, si las hay. A continuación, la aplicación debe convertir el objeto `AWSCredentials` en un objeto [SSOAWSCredentials](#) y establecer algunas propiedades en [Opciones](#), incluido el método de devolución de llamada que se usará para solicitar al usuario la información de inicio de sesión, si es necesario.

Este método se muestra en el siguiente fragmento de código.

Important

La aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución del SSO funcione:

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

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

```
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");

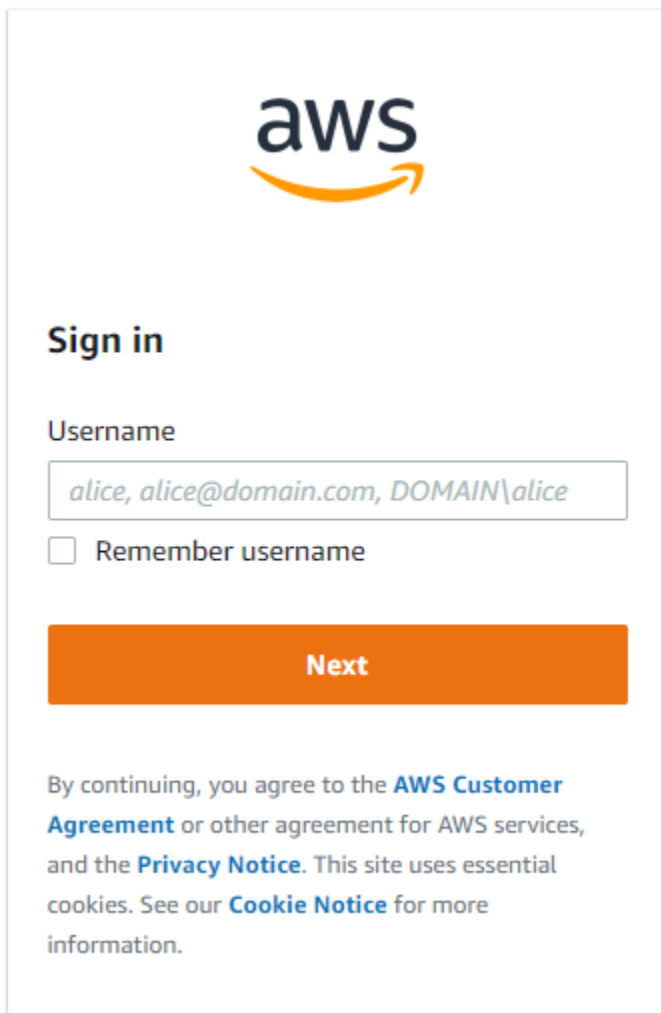
    var ssoCredentials = credentials as SSOAWSCredentials;

    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
    });
};
ssoCredentials.Options.SupportsGettingNewToken = true;

return ssoCredentials;
}
```

Si no hay un token de inicio de sesión único apropiado disponible, se abre la ventana del navegador predeterminada y la página de inicio de sesión correspondiente. Por ejemplo, si utiliza IAM Identity Center como origen de identidad, el usuario verá una página de inicio de sesión similar a la siguiente:



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

Note

La cadena de texto de `SSOAWSCredentials.Options.ClientName` que proporcione no puede tener espacios. Si la cadena tiene espacios, se producirá una excepción de tiempo de ejecución.

[Tutorial de SSO utilizando únicamente aplicaciones .NET](#)

AWS CLI y una aplicación.NET

En esta sección, se muestra cómo generar un token de SSO temporal mediante el AWS CLI token y cómo usarlo en una aplicación. Para ver un tutorial completo de este proceso, consulte [Tutorial para el inicio de sesión único mediante AWS CLI las aplicaciones y.NET](#).

Genera un token de SSO mediante el AWS CLI

Además de generar un token de SSO temporal mediante programación, se utiliza AWS CLI para generar el token. La siguiente información describe cómo hacerlo.

Una vez que el usuario crea un perfil habilitado para SSO, como se muestra en la [sección anterior](#), este ejecuta el comando `aws sso login` desde AWS CLI. Debe asegurarse de incluir el parámetro `--profile` con el nombre del perfil habilitado para SSO. Esto se muestra en el siguiente ejemplo:

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

Si el usuario quiere generar un nuevo token temporal después de que caduque el actual, puede volver a ejecutar el mismo comando.

Uso del token de SSO generado en una aplicación .NET

La siguiente información muestra cómo utilizar un token temporal que ya se ha generado.

Important

Tu aplicación debe hacer referencia a los siguientes NuGet paquetes para que la resolución del SSO funcione:

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

Si no se hace referencia a estos paquetes, se producirá una excepción de tiempo de ejecución.

La aplicación crea un objeto [AWSCredentials](#) para el perfil de SSO, que carga las credenciales temporales generadas anteriormente por AWS CLI. Esto es similar a los métodos que se muestran en [Acceso a las credenciales y perfiles en una aplicación](#) y presenta el siguiente 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;  
}
```

A continuación, el objeto `AWSCredentials` se pasa al constructor de un cliente de servicio. Por ejemplo:

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

Note

No es necesario usar `AWSCredentials` para cargar credenciales temporales si la aplicación se ha creado para usar el perfil [default] para SSO. En ese caso, la aplicación puede crear clientes de AWS servicio sin parámetros, de forma similar a "var client = new AmazonS3Client();».

[Tutorial para el inicio de sesión único mediante AWS CLI las aplicaciones y.NET](#)

Recursos adicionales

Para obtener más ayuda, consulte los recursos siguientes:

- [¿Qué es IAM Identity Center?](#)
- [Configurar el IAM Identity Center AWS CLI para usar](#)
- [Uso de las credenciales del Centro de Identidad de IAM en el AWS Toolkit for Visual Studio](#)

Tutoriales

Temas

- [Tutorial de SSO utilizando únicamente aplicaciones .NET](#)
- [Tutorial para el inicio de sesión único mediante AWS CLI las aplicaciones y.NET](#)

Tutorial de SSO utilizando únicamente aplicaciones .NET

En este tutorial se explica cómo habilitar SSO para una aplicación básica y un usuario de SSO de prueba. En él se configura la aplicación para generar un token de SSO temporal mediante programación, en lugar de [utilizar AWS CLI](#).

En este tutorial, se muestra una pequeña parte de la funcionalidad de SSO en. AWS SDK para .NET. Para obtener más información sobre el uso del Centro de identidad de IAM con el AWS SDK para .NET, consulte el tema con información general. En ese tema, consulte en particular la descripción general de este escenario en la subsección [Solo aplicación .NET](#).

Note

Varios de los pasos de este tutorial le ayudan a configurar servicios como AWS Organizations el Centro de identidad de IAM. Si ya ha realizado esa configuración, o si solo le interesa el código, puede pasar a la sección con el [código de ejemplo](#).

Requisitos previos

- Configure el entorno de desarrollo si aún no lo ha hecho. Esto se describe en secciones como [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#) y [Introducción](#).
- Identifique o cree al menos una Cuenta de AWS que pueda usar para probar el SSO. En este tutorial, esta cuenta se denomina Cuenta de AWS de prueba o, simplemente, cuenta de prueba.
- Identifique un usuario de SSO que pueda probar el SSO en su nombre. Será la persona que use el SSO y las aplicaciones básicas que cree. En este tutorial, esa persona puede ser usted (el desarrollador) u otra persona. También recomendamos una configuración en la que el usuario de SSO trabaje en un equipo que no esté en el entorno de desarrollo. Sin embargo, esto no es estrictamente necesario.
- El equipo del usuario de SSO debe tener instalada una versión de .NET Framework compatible con la que se ha utilizado para configurar el entorno de desarrollo.

Configura AWS

En esta sección se muestra cómo configurar varios AWS servicios para este tutorial.

Para realizar esta configuración, primero inicie sesión en la prueba Cuenta de AWS como administrador. A continuación, proceda del modo siguiente:

Amazon S3

Vaya a la [consola Amazon S3](#) y agregue unos cuantos buckets inocuos. Más adelante en este tutorial, el usuario de SSO recuperará una lista de estos buckets.

AWS YO SOY

Vaya a la [consola de IAM](#) y agregue unos cuantos usuarios de IAM. Si concede permisos a los usuarios de IAM, limite estos permisos a unos pocos permisos de solo lectura inocuos. Más adelante en este tutorial, el usuario de SSO recuperará una lista de estos usuarios de IAM.

AWS Organizations

Vaya a la [consola de AWS Organizations](#) y active Organizations. Para obtener más información, consulte [Creación de una organización](#) en la [Guía del usuario de AWS Organizations](#).

Esta acción agrega la prueba Cuenta de AWS a la organización como cuenta de administración. Si tiene más cuentas de prueba, puede invitarlas a unirse a la organización, pero en este tutorial no es necesario hacerlo.

IAM Identity Center

Vaya a la [consola de IAM Identity Center](#) y active el inicio de sesión único. Realice la verificación de correo electrónico si es necesario. Para obtener más información, consulte [¿Qué es IAM Identity Center?](#) en la [Guía del usuario de IAM Identity Center](#).

A continuación, realice la siguiente configuración.

Configurar IAM Identity Center

1. Vaya a la página Configuración. Busque la “URL de portal de acceso” y registre el valor para usarlo más adelante en el ajuste `sso_start_url`.
2. En el encabezado de Consola de administración de AWS, busca la Región de AWS que se configuró cuando habilitaste el SSO. Este es el menú desplegable a la izquierda del Cuenta de AWS ID. Registre el código de región para usarlo más adelante en el ajuste `sso_region`. Este código será similar a `us-east-1`.
3. Cree un usuario de SSO de la siguiente manera:
 - a. Vaya a la página Usuarios.
 - b. Seleccione Agregar usuario e introduzca el nombre de usuario, la dirección de correo electrónico, el nombre y el apellido del usuario. A continuación, elija Siguiente.
 - c. Seleccione Siguiente en la página de grupos, revise la información y seleccione Agregar usuario.
4. Cree un grupo de la siguiente manera:

- a. Vaya a la página Grupos.
 - b. Seleccione Crear grupo e introduzca el nombre de grupo y la descripción del grupo.
 - c. En la sección Agregar usuarios al grupo, seleccione el usuario de SSO de prueba que creó anteriormente. Seleccione Crear grupo.
5. Cree un conjunto de permisos de la siguiente manera:
- a. Vaya a la página Conjuntos de permisos y seleccione Crear conjunto de permisos.
 - b. En Tipo de conjunto de permisos, seleccione Conjunto de permisos personalizado y seleccione Siguiente.
 - c. Abra Política insertada e introduzca la siguiente política:

JSON

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

- d. En este tutorial, introduzca `SSOReadOnlyRole` como nombre del conjunto de permisos. Agregue una descripción si lo desea y, a continuación, seleccione Siguiente.
 - e. Revise la información y seleccione Crear.
 - f. Registre el nombre del conjunto de permisos para usarlo más adelante en el ajuste `sso_role_name`.
6. Ve a la página de AWS cuentas y elige la AWS cuenta que agregaste anteriormente a la organización.
7. En la sección Información general de esa página, busque el ID de la cuenta y regístrelo para usarlo más adelante en el ajuste `sso_account_id`.

8. Seleccione la pestaña Usuarios y grupos y, a continuación, seleccione Asignar usuarios y grupos.
9. En la página Asignar usuarios y grupos, seleccione la pestaña Grupos, luego el grupo que creó anteriormente y, por último, Siguiente.
10. Seleccione el conjunto de permisos que creó anteriormente, luego Siguiente y, por último, Enviar. La configuración tardará un poco en surtir efecto.

Creación de aplicaciones de ejemplo

Cree las siguientes aplicaciones. Se ejecutarán en el equipo del usuario de SSO.

Enumerar buckets de Amazon S3

Incluye NuGet paquetes `AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO` y `AWSSDK.SSO0IDC` además de `AWSSDK.S3` y `AWSSDK.SecurityToken`.

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

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

namespace SS0Example.S3.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An 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($"SSO 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");

        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
    });
};
ssoCredentials.Options.SupportsGettingNewToken = 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;
    }
}
}
}

```

Enumerar usuarios de IAM

Incluya NuGet paquetes `AWSSDK.IdentityManagement` y `AWSSDK.SecurityToken` además de `AWSSDK.SSO` y `AWSSDK.SSO0IDC`.

```

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
            });
        };
        ssoCredentials.Options.SupportsGettingNewToken = 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;
    }
}
}

```

Aparte de enumerar los buckets de Amazon S3 y los usuarios de IAM, estas aplicaciones muestran el ARN de identidad de usuario del perfil habilitado para SSO, que en este tutorial es `my-sso-profile`.

Estas aplicaciones realizan tareas de inicio de sesión único al proporcionar un método de devolución de llamada en la propiedad [Options](#) de un objeto. [SSOAWSCredentials](#)

Indicaciones para el usuario de SSO

Pida al usuario de SSO que consulte su correo electrónico y acepte la invitación de SSO. Se le pedirá que establezca una contraseña. Es posible que el mensaje tarde unos minutos en llegar a la bandeja de entrada del usuario de SSO.

Proporcione al usuario de SSO las aplicaciones que creó anteriormente.

A continuación, pida al usuario de SSO que haga lo siguiente:

1. Si la carpeta que contiene el AWS config archivo compartido no existe, créela. Si la carpeta existe y tiene una subcarpeta llamada `.sso`, elimínela.

La ubicación de esta carpeta suele ser `%USERPROFILE%\ .aws` en Windows y `~/ .aws` en Linux y macOS.

2. Cree un AWS config archivo compartido en esa carpeta, si es necesario, y agréguele un perfil de la siguiente manera:

```
[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. Ejecute la aplicación Amazon S3.
4. Inicie sesión en la página de inicio de sesión web que aparece. Use el nombre de usuario del mensaje de invitación y la contraseña que se creó en respuesta al mensaje.
5. Cuando el inicio de sesión se complete, la aplicación mostrará la lista de buckets de S3.
6. Ejecute la aplicación IAM. La aplicación muestra la lista de usuarios de IAM. Esto sucede aun cuando no se haya realizado un segundo inicio de sesión. La aplicación IAM usa el token temporal que se creó anteriormente.

Limpieza

Si no quiere conservar los recursos que ha creado durante este tutorial, límpielos. Pueden ser AWS recursos o recursos de su entorno de desarrollo, como archivos y carpetas.

Tutorial para el inicio de sesión único mediante AWS CLI las aplicaciones y.NET

En este tutorial se explica cómo habilitar SSO para una aplicación .NET básica y un usuario de SSO de prueba. Utiliza el AWS CLI para generar un token de SSO temporal en lugar de [generarlo mediante programación](#).

En este tutorial, se muestra una pequeña parte de la funcionalidad de SSO del AWS SDK para .NET. Para obtener más información sobre el uso del Centro de identidad de IAM con el AWS SDK para .NET, consulte el tema con información general. En ese tema, consulte en particular la descripción general de este escenario en la subsección [AWS CLI y una aplicación.NET](#).

Note

Varios de los pasos de este tutorial le ayudan a configurar servicios como AWS Organizations el Centro de identidad de IAM. Si ya ha realizado estas configuraciones, o si solo le interesa el código, puede pasar a la sección con el [código de ejemplo](#).

Requisitos previos

- Configure el entorno de desarrollo si aún no lo ha hecho. Esto se describe en secciones como [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#) y [Introducción](#).
- Identifique o cree al menos una Cuenta de AWS que pueda usar para probar el SSO. En este tutorial, esta cuenta se denomina Cuenta de AWS de prueba o, simplemente, cuenta de prueba.
- Identifique un usuario de SSO que pueda probar el SSO en su nombre. Será la persona que use el SSO y las aplicaciones básicas que cree. En este tutorial, esa persona puede ser usted (el desarrollador) u otra persona. También recomendamos una configuración en la que el usuario de SSO trabaje en un equipo que no esté en el entorno de desarrollo. Sin embargo, esto no es estrictamente necesario.
- El equipo del usuario de SSO debe tener instalada una versión de .NET Framework compatible con la que se ha utilizado para configurar el entorno de desarrollo.
- Asegúrese de que la AWS CLI versión 2 esté [instalada](#) en el ordenador del usuario del SSO. Para comprobarlo, ejecute `aws --version` en una línea de comandos o en un terminal.

Configurar AWS

En esta sección se muestra cómo configurar varios AWS servicios para este tutorial.

Para realizar esta configuración, primero inicie sesión en la prueba Cuenta de AWS como administrador. A continuación, proceda del modo siguiente:

Amazon S3

Vaya a la [consola Amazon S3](#) y agregue unos cuantos buckets inocuos. Más adelante en este tutorial, el usuario de SSO recuperará una lista de estos buckets.

AWS YO SOY

Vaya a la [consola de IAM](#) y agregue unos cuantos usuarios de IAM. Si concede permisos a los usuarios de IAM, limite estos permisos a unos pocos permisos de solo lectura inocuos. Más adelante en este tutorial, el usuario de SSO recuperará una lista de estos usuarios de IAM.

AWS Organizations

Vaya a la [consola de AWS Organizations](#) y active Organizations. Para obtener más información, consulte [Creación de una organización](#) en la [Guía del usuario de AWS Organizations](#).

Esta acción agrega la prueba Cuenta de AWS a la organización como cuenta de administración. Si tiene más cuentas de prueba, puede invitarlas a unirse a la organización, pero en este tutorial no es necesario hacerlo.

IAM Identity Center

Vaya a la [consola de IAM Identity Center](#) y active el inicio de sesión único. Realice la verificación de correo electrónico si es necesario. Para obtener más información, consulte [¿Qué es IAM Identity Center?](#) en la [Guía del usuario de IAM Identity Center](#).

A continuación, realice la siguiente configuración.

Configurar IAM Identity Center

1. Vaya a la página Configuración. Busque la “URL de portal de acceso” y registre el valor para usarlo más adelante en el ajuste `sso_start_url`.
2. En el encabezado de Consola de administración de AWS, busca la Región de AWS que se configuró cuando habilitaste el SSO. Este es el menú desplegable a la izquierda del Cuenta de

AWS ID. Registre el código de región para usarlo más adelante en el ajuste `sso_region`. Este código será similar a `us-east-1`.

3. Cree un usuario de SSO de la siguiente manera:
 - a. Vaya a la página Usuarios.
 - b. Seleccione Agregar usuario e introduzca el nombre de usuario, la dirección de correo electrónico, el nombre y el apellido del usuario. A continuación, elija Siguiente.
 - c. Seleccione Siguiente en la página de grupos, revise la información y seleccione Agregar usuario.
4. Cree un grupo de la siguiente manera:
 - a. Vaya a la página Grupos.
 - b. Seleccione Crear grupo e introduzca el nombre de grupo y la descripción del grupo.
 - c. En la sección Agregar usuarios al grupo, seleccione el usuario de SSO de prueba que creó anteriormente. Seleccione Crear grupo.
5. Cree un conjunto de permisos de la siguiente manera:
 - a. Vaya a la página Conjuntos de permisos y seleccione Crear conjunto de permisos.
 - b. En Tipo de conjunto de permisos, seleccione Conjunto de permisos personalizado y seleccione Siguiente.
 - c. Abra Política insertada e introduzca la siguiente política:

JSON

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

```
}
```

- d. En este tutorial, introduzca `SSOReadOnlyRole` como nombre del conjunto de permisos. Agregue una descripción si lo desea y, a continuación, seleccione **Siguiente**.
 - e. Revise la información y seleccione **Crear**.
 - f. Registre el nombre del conjunto de permisos para usarlo más adelante en el ajuste `sso_role_name`.
6. Ve a la página de AWS cuentas y elige la AWS cuenta que agregaste anteriormente a la organización.
 7. En la sección Información general de esa página, busque el ID de la cuenta y regístrelo para usarlo más adelante en el ajuste `sso_account_id`.
 8. Seleccione la pestaña Usuarios y grupos y, a continuación, seleccione **Asignar usuarios y grupos**.
 9. En la página Asignar usuarios y grupos, seleccione la pestaña Grupos, luego el grupo que creó anteriormente y, por último, **Siguiente**.
 10. Seleccione el conjunto de permisos que creó anteriormente, luego **Siguiente** y, por último, **Enviar**. La configuración tardará un poco en surtir efecto.

Creación de aplicaciones de ejemplo

Cree las siguientes aplicaciones. Se ejecutarán en el equipo del usuario de SSO.

Enumerar buckets de Amazon S3

Incluye NuGet paquetes `AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO` y `AWSSDK.SSO0IDC` además de `AWSSDK.S3` y `AWSSDK.SecurityToken`.

```
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 SSOExample.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;
    }
}
}

```

Enumerar usuarios de IAM

Incluya NuGet paquetes `AWSSDK.SSO` y `AWSSDK.SSO0IDC` además de `AWSSDK.IdentityManagement` y `AWSSDK.SecurityToken`.

```

using System;
using System.Threading.Tasks;

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

namespace 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;
}
}
```

Aparte de enumerar los buckets de Amazon S3 y los usuarios de IAM, estas aplicaciones muestran el ARN de identidad de usuario del perfil habilitado para SSO, que en este tutorial es `my-sso-profile`.

Indicaciones para el usuario de SSO

Pida al usuario de SSO que consulte su correo electrónico y acepte la invitación de SSO. Se le pedirá que establezca una contraseña. Es posible que el mensaje tarde unos minutos en llegar a la bandeja de entrada del usuario de SSO.

Proporcione al usuario de SSO las aplicaciones que creó anteriormente.

A continuación, pida al usuario de SSO que haga lo siguiente:

1. Si la carpeta que contiene el AWS config archivo compartido no existe, créela. Si la carpeta existe y tiene una subcarpeta llamada `.sso`, elimínela.

La ubicación de esta carpeta suele ser `%USERPROFILE%\ .aws` en Windows y `~/ .aws` en Linux y macOS.

2. Cree un AWS config archivo compartido en esa carpeta, si es necesario, y agréguele un perfil de la siguiente manera:

```
[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. Ejecute la aplicación Amazon S3. Aparece una excepción de tiempo de ejecución.
4. Ejecute el siguiente AWS CLI comando:

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

5. Inicie sesión en la página de inicio de sesión web que aparece. Use el nombre de usuario del mensaje de invitación y la contraseña que se creó en respuesta al mensaje.
6. Vuelva a ejecutar la aplicación Amazon S3. La aplicación muestra ahora la lista de buckets de S3.
7. Ejecute la aplicación IAM. La aplicación muestra la lista de usuarios de IAM. Esto sucede aun cuando no se haya realizado un segundo inicio de sesión. La aplicación IAM usa el token temporal que se creó anteriormente.

Limpieza

Si no quiere conservar los recursos que ha creado durante este tutorial, límpielos. Pueden ser AWS recursos o recursos de su entorno de desarrollo, como archivos y carpetas.

Implemente aplicaciones en AWS

Una vez desarrollada la aplicación o el servicio de .NET Core nativo en la nube en un equipo de desarrollo, conviene implementarlo en AWS. Puede hacerlo utilizando el Consola de administración de AWS o ciertos servicios, como CloudFormation o AWS Cloud Development Kit (AWS CDK). También puede usar AWS herramientas que se han creado con el propósito de implementarlas. Con estas herramientas se puede hacer lo siguiente.

Implementación desde la CLI de .NET

Puede usar las siguientes AWS herramientas de la CLI de .NET para implementar sus aplicaciones en AWS:

- [AWS Herramienta de implementación para .NET CLI](#): admite implementaciones en [Amazon Elastic Container Service \(Amazon ECS\)](#) y [AWS App Runner](#) [AWS Elastic Beanstalk](#)
- [AWS Lambda Herramientas para la CLI de .NET](#): admite la implementación de AWS Lambda proyectos.

Implementación desde kits de herramientas de IDE

Puede utilizar los AWS kits de herramientas para implementar sus aplicaciones directamente desde el IDE que prefiera:

- [AWS Toolkit for Visual Studio](#)

Note

La función «Publicar en AWS» del kit de herramientas presenta la misma funcionalidad que la herramienta de AWS implementación para .NET CLI. Para obtener más información, consulte [Publicación en AWS](#) en la Guía del usuario de AWS Toolkit for Visual Studio .

- [AWS Toolkit for JetBrains](#)

[Consulte Trabajar con aplicaciones AWS sin servidor y Trabajar con. AWS App Runner](#)

- [Kit de herramientas de AWS para VS Code](#)

Consulte [Uso de aplicaciones sin servidor](#) y [Uso de AWS App Runner](#).

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

Casos de uso

Las siguientes secciones contienen escenarios de casos de uso para ciertos tipos de aplicaciones, incluida información sobre cómo se usaría la CLI de .NET para implementar esas aplicaciones.

- [Aplicaciones ASP.NET Core](#)
- [Aplicaciones de consola de .NET](#)
- [Aplicaciones Blazor WebAssembly](#)
- [AWS Lambda proyectos](#)

Aplicaciones ASP.NET Core

La [Herramienta de implementación de AWS](#) de la CLI de .NET le ayuda a implementar aplicaciones ASP.NET y le guía a través del proceso de implementación. Es una herramienta interactiva de la CLI de .NET que ayuda a implementar aplicaciones .NET teniendo nociones mínimas de AWS .

La Herramienta de implementación tiene las siguientes capacidades:

- Recomendaciones de procesamiento para su aplicación: obtenga las recomendaciones de procesamiento y descubra qué AWS procesamiento es el más adecuado para su aplicación.
- Generación de Dockerfile: la herramienta genera un Dockerfile si es necesario o utiliza uno ya existente.
- Empaquetado e implementación automáticos: la herramienta crea los artefactos de implementación, aprovisiona la infraestructura mediante un proyecto de AWS CDK implementación generado e implementa la aplicación en la AWS computación elegida.
- Implementaciones repetibles y compartibles: puede generar y modificar proyectos de AWS CDK implementación para adaptarlos a su caso de uso específico. También puede controlar las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.
- Ayuda AWS CDK para aprender sobre .NET: la herramienta le ayuda a aprender gradualmente las AWS herramientas subyacentes en las que se basa, como la. AWS CDK

La [AWS herramienta](#) de implementación permite implementar aplicaciones de ASP.NET Core en los siguientes AWS servicios:

- Uso del [servicio Amazon ECS AWS Fargate](#): admite el despliegue de aplicaciones web en Amazon Elastic Container Service (Amazon ECS) con una potencia informática gestionada por AWS Fargate un motor de cálculo sin servidor.
- [AWS App Runner](#)- Soporta los despliegues en un servicio totalmente gestionado que facilita a los desarrolladores la implementación de aplicaciones web en contenedores y a escala. APIs No se requiere experiencia previa en infraestructura.
- [AWS Elastic Beanstalk](#)- Soporta los despliegues en un servicio que facilita a los desarrolladores la implementación de aplicaciones web y en un entorno totalmente gestionado APIs a escala. No se requiere experiencia previa en infraestructura.

Para obtener más información, consulte la [descripción general de la herramienta](#). Para comenzar desde ahí, vaya a Documentación > Introducción y seleccione [Cómo instalar](#) para obtener las instrucciones de instalación.

Aplicaciones de consola de .NET

La [Herramienta de implementación de AWS](#) de la CLI de .NET le ayuda a implementar aplicaciones de consola de .NET como un servicio o una tarea programada como una imagen de contenedor en Linux, y le guía a través del proceso de implementación. Si la aplicación no tiene un Dockerfile, la herramienta lo genera automáticamente. Si no, se utiliza un Dockerfile existente.

La Herramienta de implementación tiene las siguientes capacidades:

- Recomendaciones de procesamiento para su aplicación: obtenga las recomendaciones de procesamiento y descubra qué AWS procesamiento es el más adecuado para su aplicación.
- Generación de Dockerfile: la herramienta genera un Dockerfile si es necesario o utiliza uno ya existente.
- Empaquetado e implementación automáticos: la herramienta crea los artefactos de implementación, aprovisiona la infraestructura mediante un proyecto de AWS CDK implementación generado e implementa la aplicación en la AWS computación elegida.
- Implementaciones repetibles y compartibles: puede generar y modificar proyectos de AWS CDK implementación para adaptarlos a su caso de uso específico. También puede controlar

las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.

- Ayuda AWS CDK para aprender sobre .NET: la herramienta le ayuda a aprender gradualmente las AWS herramientas subyacentes en las que se basa, como la. AWS CDK

La [AWS herramienta](#) de implementación permite implementar aplicaciones de consola .NET en los siguientes AWS servicios:

- Uso del [servicio Amazon ECS AWS Fargate](#): admite el despliegue de aplicaciones de.NET como un servicio (por ejemplo, un procesador en segundo plano) en Amazon Elastic Container Service (Amazon ECS) con una potencia de cálculo gestionada AWS Fargate por un motor de cálculo sin servidor.
- Uso de [tareas programadas de Amazon ECS AWS Fargate](#): admite la implementación de aplicaciones de.NET como una tarea programada (por ejemplo, un end-of-day proceso) en Amazon ECS con una potencia de procesamiento administrada por un motor de procesamiento AWS Fargate sin servidor.

Para obtener más información, consulte la [descripción general de la herramienta](#). Para comenzar desde ahí, vaya a Documentación > Introducción y seleccione [Cómo instalar](#) para obtener las instrucciones de instalación.

Aplicaciones Blazor WebAssembly

La [herramienta de AWS implementación](#) para la CLI de.NET le ayuda a alojar su WebAssembly aplicación Blazor en Amazon S3, utilizando Amazon CloudFront para la entrega de contenido a la red. La aplicación se implementa en un bucket de S3 para su alojamiento web. La herramienta crea y configura un bucket de S3 y, a continuación, sube la aplicación Blazor a dicho bucket.

La Herramienta de implementación tiene las siguientes capacidades:

- Empaquetado e implementación automáticos: la herramienta crea los artefactos de implementación, aprovisiona la infraestructura mediante un proyecto de AWS CDK implementación generado e implementa la aplicación en el cómputo elegido AWS .
- Implementaciones repetibles y compartibles: puede generar y modificar proyectos de AWS CDK implementación para adaptarlos a su caso de uso específico. También puede controlar las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.

- Ayuda AWS CDK para aprender sobre .NET: la herramienta le ayuda a aprender gradualmente las AWS herramientas subyacentes en las que se basa, como la. AWS CDK

Para obtener más información, consulte la [descripción general de la herramienta](#). Para comenzar desde ahí, vaya a Documentación > Introducción y seleccione [Cómo instalar](#) para obtener las instrucciones de instalación.

AWS Lambda proyectos

AWS Lambda es un servicio informático que le permite ejecutar código sin aprovisionar ni administrar servidores. Ejecuta su código en una infraestructura informática de alta disponibilidad y realiza todas las tareas de administración de los recursos informáticos. Para obtener más información sobre Lambda, consulte [¿Qué es AWS Lambda?](#) en la Guía para AWS Lambda desarrolladores.

Puede implementar funciones de Lambda usando la interfaz de la línea de comandos (CLI) de .NET.

Temas

- [Requisitos previos](#)
- [Comandos de Lambda disponibles](#)
- [Pasos de la implementación](#)

Requisitos previos

Para empezar a utilizar la CLI de .NET para implementar funciones de Lambda, debe cumplir los siguientes requisitos previos:

- Confirme que tiene instalada la CLI de .NET. Por ejemplo: `dotnet --version`. Si es necesario, vaya a <https://dotnet.microsoft.com/download> para instalarlo.
- Configure la CLI de .NET para que funcione con Lambda. Para obtener una descripción de cómo hacerlo, consulte [CLI de .NET Core](#) en la Guía para desarrolladores de AWS Lambda . En ese procedimiento, el comando de implementación es este:

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

Si no está seguro de cómo crear un rol de IAM en este ejercicio, no incluya la parte `--function-role role`. La herramienta le ayudará a crear uno nuevo.

Comandos de Lambda disponibles

Para ver una lista de los comandos de Lambda que están disponibles a través de la CLI de .NET, abra una línea de comandos o una terminal y escriba `dotnet lambda --help`. El resultado del comando será similar a lo siguiente:

```
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>
```

El resultado muestra todos los comandos que están disponibles actualmente.

Pasos de la implementación

En las instrucciones siguientes se supone que ha creado un proyecto AWS Lambda de .NET. En este procedimiento, el proyecto se llama `DotNetCoreLambdaTest`.

1. Abra un símbolo del sistema o terminal y vaya a la carpeta que contiene archivo del proyecto de .NET de Lambda.
2. Escriba `dotnet lambda deploy-function`.
3. Si se le solicita, introduzca la AWS región (la región en la que se desplegará la función Lambda).
4. Cuando se le pida, escriba el nombre de la función que va a implementar, por ejemplo, `DotNetCoreLambdaTest`. Puede ser el nombre de una función que ya existe en la Cuenta de AWS o de una función que aún no está implementada ahí.
5. Cuando se le pida, seleccione o cree el rol de IAM que Lambda asumirá al ejecutar la función.

Cuando la ejecución finaliza correctamente, se muestra el mensaje Se ha creado una nueva función de Lambda.

```
Executing publish command
```

```
...  
(etc.)  
New Lambda function created
```

Si implementa una función que ya existe en la cuenta, la función de implementación solo pide la región de AWS (si procede). En tal caso, el resultado del comando termina con `Updating code for existing function`.

Una vez que la función de Lambda se haya implementado, estará lista para su uso. Para obtener más información, consulte [ejemplos sobre cómo usar AWS Lambda](#).

Lambda supervisa automáticamente las funciones de Lambda por usted e informa de las métricas a través de Amazon CloudWatch. Para monitorizar la función de Lambda y solucionar problemas, consulte [Supervisión y solución de problemas de funciones de Lambda](#).

AWS Servicios de llamadas desde AWS SDK para .NET

Las siguientes secciones contienen ejemplos, tutoriales, tareas y guías que muestran cómo utilizarlos AWS SDK para .NET para trabajar con AWS los servicios. Estos ejemplos y tutoriales se basan en una API que el AWS SDK para .NET proporciona. Para ver qué clases y métodos están disponibles en la API, consulte la [Referencia de API de AWS SDK para .NET](#).

Si es la primera vez que lo conoces AWS SDK para .NET, quizás quieras consultar primero el [Crear una aplicación sencilla](#) tema. que sirve de introducción a SDK.

Puedes encontrar más ejemplos de código en el repositorio de [ejemplos de AWS código y en el repositorio awslabs](#) de. GitHub

Antes de empezar, asegúrese de haber [configurado su entorno y su](#) proyecto. Revise también la información disponible en [Uso del SDK](#).

Temas

- [Ejemplos de código guiados para el AWS SDK para .NET](#)
- [Uso AWS Lambda para servicio de cómputo](#)
- [Bibliotecas y marcos de alto nivel para AWS SDK para .NET](#)
- [Support para otros AWS servicios y configuraciones](#)

Ejemplos de código guiados para el AWS SDK para .NET

Las siguientes secciones contienen ejemplos de código y proporcionan orientación sobre esos ejemplos. Pueden ayudarlo a aprender a usarlo AWS SDK para .NET para trabajar con AWS los servicios. Para ver otros ejemplos de código, consulte [Ejemplos de código](#).

Note

También están disponibles ejemplos de código específicos de la versión AWS SDK para .NET 3 del. Para encontrarlos, consulte los [ejemplos de código con orientación](#) y los [ejemplos de código](#) en la Guía para desarrolladores AWS SDK para .NET (V3). Sin embargo, ten en cuenta que si utilizas los ejemplos de código específicos de la versión 3 con la versión 4 del SDK (la versión más reciente), es posible que tengas que realizar ajustes de acuerdo con la información que se incluye en ella. [Migración a la versión 4](#)

Si es la primera vez que lo AWS SDK para .NET conoces, quizás quieras consultar primero el [Crear una aplicación sencilla](#) tema, que sirve de introducción a SDK.

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información disponible en [Uso del SDK](#).

Temas

- [Acceder CloudFormation con el AWS SDK para .NET](#)
- [Autenticación de usuarios con Amazon Cognito](#)
- [Uso de bases de datos NoSQL de Amazon DynamoDB](#)
- [Uso de Amazon EC2](#)
- [Acceder AWS Identity and Access Management \(IAM\) con el AWS SDK para .NET](#)
- [Uso del servicio de almacenamiento de Internet de Amazon Simple Storage Service](#)
- [Envío de notificaciones desde la nube mediante Amazon Simple Notification Service](#)
- [Mensajería mediante Amazon SQS](#)

Acceder CloudFormation con el AWS SDK para .NET

El AWS SDK para .NET soporte [AWS CloudFormation](#), que crea y aprovisiona los despliegues de AWS infraestructura de forma predecible y repetida.

APIs

Los servicios APIs para AWS SDK para .NET los clientes. CloudFormation Le APIs permiten trabajar con CloudFormation funciones como plantillas y pilas. Esta sección contiene una pequeña cantidad de ejemplos que le muestran los patrones que puede seguir al trabajar con ellos APIs. Para ver el conjunto completo de APIs, consulta la [referencia de la AWS SDK para .NET API](#) (y desplázate hasta «Amazon. CloudFormation»).

AWS CloudFormation APIs Son proporcionados por [AWSSDK. CloudFormation](#) paquete.

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información en [Uso del SDK](#).

Temas

Temas

- [AWS Enumere los recursos mediante AWS CloudFormation](#)

AWS Enumere los recursos mediante AWS CloudFormation

En este ejemplo, se muestra cómo utilizarlos AWS SDK para .NET para enumerar los recursos en CloudFormation pilas. En el ejemplo se utiliza la API de bajo nivel. La aplicación no toma argumentos, sino que simplemente recopila información de todas las pilas a las que se puede acceder con las credenciales del usuario y muestra información sobre esas pilas.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.CloudFormation](#)

Elementos de programación:

- [Espacio de nombres Amazon. CloudFormation](#)

Clase [AmazonCloudFormationClient](#)

- [Espacio de nombres Amazon. CloudFormation](#).Modelo

[Clase. ICloud FormationPaginatorFactory DescribeStacks](#)

Clase [DescribeStackResourcesRequest](#)

Clase [DescribeStackResourcesResponse](#)

Clase [Stack](#)

Clase [StackResource](#)

Clase [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($"In 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 SSO, be sure to install" +
            " the AWSSDK.SSO and AWSSDK.SSO0IDC 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 SSO, have you logged in?");
        }
        else
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine(ex.StackTrace);
        }

        return false;
    }
}
```

Autenticación de usuarios con Amazon Cognito

Note

La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

Con Amazon Cognito Identity, puede crear identidades únicas para sus usuarios y autenticarlos para un acceso seguro a sus AWS recursos, como Amazon S3 o Amazon DynamoDB. Amazon Cognito Identity es compatible con proveedores de identidades públicos como Amazon, Facebook, Twitter/ Digits, Google o cualquier proveedor compatible con OpenID Connect, así como con identidades sin autenticar. Cognito también admite [identidades autenticadas por el desarrollador](#), que le permiten registrar y autenticar usuarios mediante su propio proceso de autenticación backend, sin dejar de usar Amazon Cognito Sync para sincronizar los datos de los usuarios y obtener acceso a los recursos de AWS .

Para obtener más información sobre [Amazon Cognito](#), consulte la [Guía para desarrolladores de Amazon Cognito](#).

En los siguientes ejemplos de código se muestra cómo utilizar Amazon Cognito Identity fácilmente. En el ejemplo del tema [Proveedor de credenciales](#) se muestra cómo crear y autenticar identidades de usuario. El [CognitoAuthentication biblioteca de extensiones](#) ejemplo muestra cómo utilizar la biblioteca de CognitoAuthentication extensiones para autenticar grupos de usuarios de Amazon Cognito.

Temas

- [Proveedor de credenciales de Amazon Cognito](#)
- [Ejemplos de bibliotecas CognitoAuthentication de extensiones de Amazon](#)

Proveedor de credenciales de Amazon Cognito

Note

La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, que se encuentra en [AWSSDK.CognitoIdentity](#) NuGetpackage, es un objeto de credenciales que utiliza Amazon Cognito y AWS Security Token Service (AWS STS) para recuperar credenciales y realizar AWS llamadas.

El primer paso en la configuración de `CognitoAWSCredentials` es crear un “grupo de identidades”. Un grupo de identidades es un almacén de información de identidades de usuarios que es específico de una cuenta determinada. La información se puede recuperar en plataformas, dispositivos y sistemas operativos cliente, de modo que si un usuario comienza a usar la aplicación en un teléfono y más tarde pasa a una tablet, la información de la aplicación persistente seguirá estando disponible para ese usuario. Puede crear un nuevo grupo de identidades desde la consola de Amazon Cognito. Si usa la consola, esta también le proporcionará los demás datos que se requieren:

- Su número de cuenta: un número de 12 dígitos, por ejemplo, 123456789012, exclusivo de su cuenta.
- El ARN de la función sin autenticación: la función que asumirán los usuarios que no se hayan autenticado. Por ejemplo, esta función puede proporcionar permisos de solo lectura respecto a los datos.

- El ARN de la función con autenticación: la función que asumirán los usuarios que se hayan autenticado. Esta función puede proporcionar permisos más extensos respecto a los datos.

Configurar Cognito AWSCredentials

En el siguiente ejemplo de código se muestra cómo configurar `CognitoAWSCredentials`, que puede usar a continuación para realizar una llamada a Amazon S3 como usuario sin autenticar. Esto permite realizar llamadas exigiendo tan solo una cantidad mínima de datos para autenticar el usuario. Los permisos de los usuarios se controlan mediante la función, lo que le permite configurar el acceso de acuerdo con sus necesidades.

```
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();
}
```

AWS Utilícelo como usuario no autenticado

El siguiente ejemplo de código muestra cómo puedes empezar a usarlo AWS como usuario no autenticado, luego autenticarte a través de Facebook y actualizar las credenciales para usar las credenciales de Facebook. Con este enfoque, puede conceder capacidades diferentes a los usuarios autenticados a través de la función con que se autenticuen. Por ejemplo, podría tener una aplicación de teléfono que permita a los usuarios ver contenido de forma anónima, pero que les permita publicarlo si inician sesión a través de uno o varios de los proveedores 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();
}
```

El objeto `CognitoAWSCredentials` proporciona más funcionalidad si cabe cuando se usa con el cliente `AmazonCognitoSyncClient` que forma parte de AWS SDK para .NET. Si usa tanto `AmazonCognitoSyncClient` como `CognitoAWSCredentials`, no es preciso especificar las propiedades `IdentityPoolId` e `IdentityId` al realizar llamadas con `AmazonCognitoSyncClient`. Estas propiedades se rellenan automáticamente desde `CognitoAWSCredentials`. Esto se ilustra en el siguiente ejemplo de código, además de un evento que le envía una notificación siempre que la propiedad `IdentityId` de `CognitoAWSCredentials` cambia. La propiedad `IdentityId` puede cambiar en algunos casos; por ejemplo, cuando un usuario sin autenticar pasa a estar 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 = "..."
    });
}
```

Ejemplos de bibliotecas CognitoAuthentication de extensiones de Amazon

Note

La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

La biblioteca CognitoAuthentication de extensiones, que se encuentra en [Amazon.Extensions.CognitoAuthentication](#) NuGet paquete, simplifica el proceso de autenticación de los grupos de usuarios de Amazon Cognito para los desarrolladores de .NET Core y Xamarin. La biblioteca se basa en la API de proveedor de identidades de Amazon Cognito para crear y enviar llamadas de API de autenticación de usuarios.

Uso de la biblioteca de extensiones CognitoAuthentication

Amazon Cognito tiene algunos valores de AuthFlow y ChallengeName integrados de flujo de autenticación estándar que validan el nombre de usuario y la contraseña mediante el protocolo Secure Remote Password (SRP). Para obtener más información acerca del flujo de autenticación, consulte [Flujo de autenticación de los grupos de usuarios](#) de Amazon Cognito.

En los siguientes ejemplos se requieren estas instrucciones 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;
```

Uso de la autenticación básica

Cree una [AmazonCognitoIdentityProviderClient](#) con [Anonymous AWSCredentials](#), que no requiere solicitudes firmadas. No es preciso proporcionar una región; el código subyacente llama a `FallbackRegionFactory.GetRegionEndpoint()` si no se proporciona una región. Cree

objetos `CognitoUserPool` y `CognitoUser`. Llame al método `StartWithSrpAuthAsync` mediante una solicitud `InitiateSrpAuthRequest` que contenga la contraseña del usuario.

```
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;
}
```

Autenticación con desafíos

Continuar con el flujo de autenticación con desafíos, como la Autenticación Multi-Factor (MFA), también es más sencillo. `NewPasswordRequired` Los únicos requisitos son los `CognitoAuthentication` objetos, la contraseña del usuario para el SRP y la información necesaria para el siguiente desafío, que se obtiene tras solicitar al usuario que la introduzca. El siguiente código muestra una forma de comprobar el tipo de desafío y obtener las respuestas adecuadas para el MFA y `NewPasswordRequired` los desafíos durante el flujo de autenticación.

Realice una solicitud de autenticación básica como antes y ejecute `await` para esperar a la respuesta `AuthFlowResponse`. Cuando haya recibido la respuesta, ejecute un bucle para el objeto `AuthenticationResult` devuelto. Si el tipo de `ChallengeName` es `NEW_PASSWORD_REQUIRED`, llame al 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.");
}
}
```

Utilice AWS los recursos después de la autenticación

Una vez que un usuario se autentica mediante la CognitoAuthentication biblioteca, el siguiente paso es permitir que el usuario acceda a los AWS recursos adecuados. Para ello, debe crear un grupo de identidades a través de la consola de identidades federadas de Amazon Cognito. Al especificar el grupo de usuarios de Amazon Cognito que creó como proveedor mediante su poolID y clientID correspondientes, puede permitir que los usuarios del grupo de usuarios de Amazon Cognito tengan acceso a los recursos de AWS conectados a su cuenta. También puede especificar diferentes funciones para que los usuarios, tanto autenticados como sin autenticar, obtengan acceso a los diferentes recursos. Puede cambiar estas reglas en la consola de IAM, donde puede agregar o eliminar permisos en el campo Acción de la política conectada al rol. A continuación, con el grupo de identidades, el grupo de usuarios y la información de usuario de Amazon Cognito adecuados, puede realizar llamadas a distintos AWS recursos. En el siguiente ejemplo se muestra un usuario autenticado con SRP que obtiene acceso a distintos buckets de Amazon S3 permitidos por el rol del grupo de identidades asociado.

```
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);
    }
}
```

Más opciones de autenticación

Además de SRP y MFA NewPasswordRequired, CognitoAuthentication la biblioteca de extensiones ofrece un flujo de autenticación más sencillo para:

- Custom: inicio con una llamada a
`StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Inicie con una llamada a
`StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP: inicie con una llamada a
`StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP: inicie con una llamada a
`StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Llame al método apropiado según el flujo que desee. A continuación, siga pidiendo al usuario que responda a los desafíos a medida que se presenten en los objetos `AuthFlowResponse` de cada llamada al método. También llame al método de respuesta adecuado; por ejemplo,

`RespondToSmsMfaAuthAsync` para los desafíos de MFA y `RespondToCustomAuthAsync` para los desafíos personalizados.

Uso de bases de datos NoSQL de Amazon DynamoDB

Note

Los modelos de programación de estos temas están presentes tanto en .NET Framework como en .NET (Core), pero las convenciones de llamada difieren entre sincrónicas y asincrónicas.

AWS SDK para .NET Es compatible con Amazon DynamoDB, que es un servicio rápido de base de datos NoSQL ofrecido por. AWS SDK proporciona tres modelos de programación para comunicarse con DynamoDB: el modelo de bajo nivel, el modelo de documento y el modelo de persistencia de objetos.

La siguiente información presenta estos modelos y sus APIs componentes, proporciona ejemplos de cómo y cuándo usarlos y proporciona enlaces a recursos de programación adicionales de DynamoDB en. AWS SDK para .NET

Modelo de bajo nivel

El modelo de programación de bajo nivel incluye llamadas directas al servicio DynamoDB. Puede acceder a este modelo a través del espacio de nombres [DBv2Amazon.Dynamo](#).

De los tres modelos, el de bajo nivel requiere que escriba la mayor parte del código. Por ejemplo, debe convertir los tipos de datos de .NET en sus equivalentes en DynamoDB. Sin embargo, este modelo le ofrece acceso a la mayoría de las características.

En el siguiente ejemplo se muestra cómo usar el modelo de bajo nivel para crear una tabla, modificarla e insertar elementos en ella en DynamoDB.

Creación de una tabla

En el siguiente ejemplo, puede crear una tabla mediante el método `CreateTable` de la clase `AmazonDynamoDBClient`. El método `CreateTable` usa una instancia de la clase `CreateTableRequest` que contiene características como los nombres de atributos de elemento obligatorios, la definición de clave principal y la capacidad de desempeño. El método `CreateTable` devuelve una instancia de la clase `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);
}

```

Verificación de la preparación de una tabla para su modificación

Antes de poder cambiar o modificar una tabla, esta debe estar lista para ello. En el siguiente ejemplo se muestra cómo usar el modelo de bajo nivel para verificar que una tabla de DynamoDB está lista. En este ejemplo, se hace referencia a la tabla de destino que se comprobará a través del método `DescribeTable` de la clase `AmazonDynamoDBClient`. Cada cinco segundos, el código comprueba el valor de la propiedad `TableStatus` de la tabla. Cuando el estado se establezca en `ACTIVE`, la tabla estará lista para modificarse.

```

// 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);
```

Inserción de un elemento en una tabla

En el siguiente ejemplo se usa el modelo de bajo nivel para insertar dos elementos en una tabla de DynamoDB. Cada elemento se inserta a través del método `PutItem` de la clase `AmazonDynamoDBClient`, mediante una instancia de la clase `PutItemRequest`. Cada una de las dos instancias de la clase `PutItemRequest` toma el nombre de la tabla en la que se insertarán los elementos, con una serie de valores de atributos de elemento.

```
// 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

El modelo de programación del documento proporciona una forma más sencilla de trabajar con datos en DynamoDB. Este modelo está destinado de manera específica al acceso a tablas y elementos de las tablas. [Puede acceder a este modelo a través de Amazon.Dynamo. DBv2 DocumentModel](#) espacio de nombres.

En comparación con el modelo de programación de bajo nivel, es más sencillo escribir el código del modelo de documento en los datos de DynamoDB. Por ejemplo, no debe convertir tantos tipos de datos de .NET en sus equivalentes en DynamoDB. Sin embargo, este modelo no proporciona acceso a tantas características como el modelo de programación de bajo nivel. Por ejemplo, puede usar este modelo para crear, recuperar, actualizar y eliminar elementos de las tablas. No obstante, para crear las tablas, debe usar el modelo de bajo nivel. En comparación con el modelo de persistencia de objetos, este modelo requiere que escriba más código para almacenar, cargar y consultar objetos de .NET.

Para obtener más información sobre el modelo de programación de documentos de DynamoDB, consulte [.NET: modelo de documento](#) en la [Guía para desarrolladores de Amazon DynamoDB](#).

En las siguientes secciones se proporciona información sobre cómo crear una representación de la tabla de DynamoDB deseada, así como ejemplos sobre cómo utilizar el modelo de documento para insertar elementos en tablas y obtener elementos de tablas.

Creación de una representación de la tabla

Para llevar a cabo operaciones de datos con el modelo de documento, primero hay que crear una instancia de la clase `Table` que representa una tabla específica. Principalmente, hay dos formas de hacer esto.

LoadTable method

El primer mecanismo consiste en utilizar uno de los métodos `LoadTable` estáticos de la clase [Table](#), similar al ejemplo siguiente:

```
var client = new AmazonDynamoDBClient();
```

```
Table table = Table.LoadTable(client, "Reply");
```

Note

Aunque este mecanismo funciona, hay veces que, bajo determinadas condiciones, puede provocar más latencias o interbloqueos debido a comportamientos de inicio en frío y de grupo de subprocesos. Para obtener más información sobre estos comportamientos, consulte la entrada de blog [Improved DynamoDB Initialization Patterns for AWS SDK para .NET](#).

TableBuilder

En la [versión 3.7.203 del TableBuilder paquete .Dynamo se introdujo un mecanismo alternativo](#), la clase `AWSSDK DBv2 NuGet`. Este mecanismo aborda los comportamientos mencionados anteriormente, ya que elimina ciertas llamadas a método implícitas, en concreto, el método `DescribeTable`. Este mecanismo se utiliza de forma similar a la del siguiente ejemplo:

```
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 obtener más información sobre este mecanismo alternativo, consulte la entrada de blog [Patrones de inicialización de DynamoDB mejorados para AWS SDK para .NET](#).

Insertión de un elemento en una tabla

En el siguiente ejemplo se inserta una respuesta en la tabla `Reply` a través del método `PutItemAsync` de la clase `Table`. El método `PutItemAsync` toma una instancia de la clase `Document`; la clase `Document` es sencillamente una colección 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);
```

Obtención de un elemento de una tabla

En el siguiente ejemplo se recupera una respuesta a través del método `GetItemAsync` de la clase `Table`. Para determinar la respuesta que se va a obtener, el `GetItemAsync` método utiliza la clave hash-and-range principal de la respuesta 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"]);
```

En el ejemplo anterior, los valores de la tabla se convierten de forma implícita en cadenas para el método `WriteLine`. Puede realizar conversiones explícitas mediante los diversos métodos `As[type]` de la clase `DynamoDBEntry`. Por ejemplo, puede convertir el valor de atributo de `Id` de forma explícita a partir de un tipo de datos `Primitive` en un GUID a través del método `AsGuid()`:

```
var guid = reply["Id"].AsGuid();
```

Modelo de persistencia de objetos

El modelo de programación de persistencia de objetos está diseñado de forma específica para almacenar, cargar y consultar objetos de .NET en DynamoDB. Puede acceder a este modelo a través de [DBv2Amazon.Dynamo. DataModel](#) espacio de nombres.

De los tres modelos, lo más sencillo es escribir el código del modelo de persistencia de objetos siempre que almacene, cargue o consulte los datos de DynamoDB. Por ejemplo, puede trabajar directamente con los tipos de datos de DynamoDB. Sin embargo, este modelo proporciona acceso únicamente a las operaciones que almacenan, cargan y consultan objetos de .NET en DynamoDB. Por ejemplo, puede usar este modelo para crear, recuperar, actualizar y eliminar elementos de las tablas. Sin embargo, primero debe crear sus tablas con el modelo de bajo nivel y, a continuación, utilizar este modelo para asignar las clases de .NET a las tablas.

Para obtener más información sobre el modelo de programación de persistencia de objetos de DynamoDB, consulte [.NET: modelo de persistencia de objetos](#) en la [Guía para desarrolladores de Amazon DynamoDB](#).

En los siguientes ejemplos se muestra cómo definir una clase de .NET que representa un elemento de DynamoDB, cómo usar una instancia de la clase de .NET para insertar un elemento de DynamoDB y cómo usar una instancia de la clase de .NET para obtener un elemento de DynamoDB de una tabla.

Definición de una clase de .NET que representa un elemento en una tabla

En el siguiente ejemplo de definición de clase, el `DynamoDBTable` atributo especifica el nombre de la tabla, mientras que los `DynamoDBRangeKey` atributos `DynamoDBHashKey` y modelan la clave hash-and-range principal de la tabla. El atributo `DynamoDBGlobalSecondaryIndexHashKey` se define de forma que se pueda crear una consulta para obtener respuestas de un autor específico.

```
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; }  
}
```

Creación de un contexto para el modelo de persistencia de objetos

Para utilizar el modelo de programación de persistencia de objetos de DynamoDB, debe crear un contexto que ofrezca una conexión a DynamoDB y que permita obtener acceso a las tablas, realizar diversas operaciones y ejecutar consultas.

Contexto básico

En el siguiente ejemplo de código se muestra cómo crear el contexto más básico.

```
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.DataModel;  
  
var client = new AmazonDynamoDBClient();  
var context = new DynamoDBContext(client);
```

Contexto con DisableFetchingTableMetadata propiedad

En el siguiente ejemplo se muestra cómo se podría establecer también la propiedad `DisableFetchingTableMetadata` de la clase `DynamoDBContextConfig` para evitar llamadas implícitas al método `DescribeTable`.

```
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.DataModel;  
  
var client = new AmazonDynamoDBClient();  
var context = new DynamoDBContext(client, new DynamoDBContextConfig  
{  
    DisableFetchingTableMetadata = true  
});
```

Si la propiedad `DisableFetchingTableMetadata` está establecida en `false` (valor predeterminado), como se muestra en el primer ejemplo, puede omitir de la clase `Reply` los atributos que describan la estructura de clave e índice de los elementos de la tabla. En su lugar, estos atributos se deducirán mediante una llamada implícita al método `DescribeTable`. Si `DisableFetchingTableMetadata` se establece en `true`, como se muestra en el segundo ejemplo, los métodos del modelo de persistencia de objetos (como `SaveAsync` y `QueryAsync`)

se basan completamente en los atributos definidos en la clase `Reply`. En tal caso, no se realiza ninguna llamada al método `DescribeTable`.

Note

Bajo determinadas condiciones, las llamadas al método `DescribeTable` pueden provocar más latencias o interbloqueos debido a comportamientos de inicio en frío y de grupo de subprocesos. Por este motivo, a veces resulta beneficioso evitar las llamadas a ese método. Para obtener más información sobre estos comportamientos, consulte la entrada de blog [Patrones de inicialización de DynamoDB mejorados para AWS SDK para .NET](#).

Uso de una instancia de la clase de .NET para insertar un elemento en una tabla

En este ejemplo, se inserta un elemento a través del método `SaveAsync` de la clase `DynamoDBContext`, que toma una instancia inicializada de la clase de .NET que representa el elemento

```
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"
});
```

Uso de una instancia de una clase de .NET para obtener elementos de una tabla

En este ejemplo, se crea una consulta para encontrar todos los registros de “Author1” mediante el método `QueryAsync` de la clase `DynamoDBContext`. Después, los elementos se recuperan mediante el método `GetNextSetAsync` de la 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);
}
```

Información adicional sobre el modelo de persistencia de objetos

Los ejemplos y las explicaciones que se muestran arriba incluyen a veces una propiedad de la clase `DynamoDBContext` llamada `DisableFetchingTableMetadata`. Esta propiedad, que se introdujo en la [versión 3.7.203 del DBv2 NuGet paquete AWSSDK .Dynamo](#), permite evitar determinadas condiciones que podrían provocar una latencia adicional o puntos muertos debido a los comportamientos de arranque en frío y de agrupación de subprocesos. Para obtener más información, consulte la entrada de blog [Patrones de inicialización de DynamoDB mejorados para AWS SDK para .NET](#).

Aquí tiene más información sobre esta propiedad.

- Esta propiedad se puede establecer globalmente en los archivos `app.config` o `web.config` si utiliza .NET Framework.

- Esta propiedad se puede establecer globalmente con la clase [AWSConfigsDynamoDB](#), como se muestra en el siguiente ejemplo.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- En algunos casos, no se pueden agregar atributos de DynamoDB a una clase de .NET; por ejemplo, si la clase está definida en una dependencia. En estos casos, se puede seguir sacando partido de la propiedad `DisableFetchingTableMetadata`. Para ello, utilice la clase [TableBuilder](#) además de la propiedad `DisableFetchingTableMetadata`. [La TableBuilder clase también se introdujo en la versión 3.7.203 del paquete .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"
});
```

Más información

Uso del AWS SDK para .NET programa DynamoDB, información y ejemplos

- [DynamoDB APIs](#)
- [Puesta en marcha de la serie DynamoDB](#)
- [Serie DynamoDB: modelo de documento](#)
- [Serie DynamoDB: esquemas de conversión](#)
- [Serie DynamoDB: modelo de persistencia de objetos](#)
- [Serie de DynamoDB: expresiones](#)
- [Uso de expresiones con Amazon DynamoDB y AWS SDK para .NET](#)
- [Compatibilidad con JSON en Amazon DynamoDB](#)

Información y ejemplos del modelo de bajo nivel

- [Trabajar con tablas mediante la API de bajo nivel AWS SDK para .NET](#)
- [Trabajar con elementos mediante la API de AWS SDK para .NET bajo nivel](#)
- [Consulta de tablas mediante la API de bajo nivel AWS SDK para .NET](#)
- [Escaneo de tablas mediante la API de bajo nivel AWS SDK para .NET](#)
- [Trabajar con índices secundarios locales mediante la API de bajo nivel AWS SDK para .NET](#)
- [Trabajar con índices secundarios globales mediante la API de bajo nivel AWS SDK para .NET](#)

Información y ejemplos del modelo de documento

- [Tipos de datos de DynamoDB](#)
- [Dinamo DBEntry](#)
- [.NET: modelo de documento](#)

Información y ejemplos del modelo de persistencia de objetos

- [.NET: modelo de persistencia de objetos](#)

Otra información útil

- Consulte [Integración AWS con .NET Aspire](#) para obtener información sobre el desarrollo con Amazon DynamoDB local a través de .NET Aspire.

Temas

- [Uso de expresiones con Amazon DynamoDB y AWS SDK para .NET](#)
- [Compatibilidad con JSON en Amazon DynamoDB](#)

Uso de expresiones con Amazon DynamoDB y AWS SDK para .NET

Note

La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

Los siguientes ejemplos de código muestran cómo utilizar el AWS SDK para .NET para programar DynamoDB con expresiones. Las expresiones se utilizan para indicar los atributos que se desea leer de un elemento en una tabla de DynamoDB. Además, se pueden utilizar al escribir un elemento para indicar las condiciones que se deben cumplir (lo que recibe el nombre también de actualización condicional) y la manera en que se deben actualizar los atributos. Algunos ejemplos de actualización sustituyen el atributo por un valor nuevo o añaden datos nuevos a una lista o mapa. Para obtener más información acerca del uso de expresiones, consulte el tema sobre [cómo leer y escribir elementos mediante expresiones](#).

Temas

- [Datos de ejemplo](#)
- [Obtención de un elemento individual utilizando expresiones y la clave principal del elemento](#)
- [Obtención de varios elementos utilizando expresiones y la clave principal de la tabla](#)
- [Obtención de varios elementos utilizando expresiones y otros atributos del elemento](#)
- [Imprimir un elemento](#)
- [Crear o reemplazar un elemento utilizando expresiones](#)
- [Actualizar un elemento utilizando expresiones](#)
- [Eliminación de un elemento utilizando expresiones](#)
- [Más información](#)

Datos de ejemplo

Los ejemplos de código de este tema hacen referencia a los dos elementos de ejemplo siguientes de una tabla de DynamoDB llamada ProductCatalog. Estos elementos describen información sobre entradas de producto en un catálogo ficticio de una tienda de bicicletas. Estos elementos se basan en el ejemplo proporcionado en [Case Study: A ProductCatalog](#) Item. Los descriptores de tipos de datos como B00L, L, M, N, NS, S y SS equivalen a los del [formato de datos 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": {
    "B00L": 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."
        ]
      }
    }
  }
}

```

```
}  
}  
}
```

Obtención de un elemento individual utilizando expresiones y la clave principal del elemento

En el siguiente ejemplo se muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` y un conjunto de expresiones para obtener y luego imprimir el elemento que tenga un `Id` 205. Solo se devuelven los siguientes atributos del elemento: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` y `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);
```

En el ejemplo anterior, la propiedad `ProjectionExpression` especifica los atributos que se deben devolver. La propiedad `ExpressionAttributeNames` especifica el marcador de posición `#pr` para que represente el atributo `ProductReviews` y el marcador de posición `#ri` para que represente el atributo `RelatedItems`. La llamada a `PrintItem` hace referencia a una función personalizada, tal como se describe en [Imprimir un elemento](#).

Obtención de varios elementos utilizando expresiones y la clave principal de la tabla

En el siguiente ejemplo se muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` y un conjunto de expresiones para obtener y luego imprimir el elemento que tenga un `Id` 301, pero únicamente si el valor de `Price` es mayor que 150. Solo se devuelven los siguientes atributos del elemento: `Id`, `Title` y todos los atributos de `ThreeStar` en `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("====");
}
```

En el ejemplo anterior, la propiedad `ProjectionExpression` especifica los atributos que se deben devolver. La propiedad `ExpressionAttributeNames` especifica el marcador de posición `#pr` para que represente el atributo `ProductReviews` y el marcador de posición `#p` para que represente el atributo `Price`. `#pr.ThreeStar` especifica que solo se devuelva el atributo `ThreeStar`. La propiedad `ExpressionAttributeValues` especifica el marcador de posición `:val` para que represente el valor 150. La propiedad `FilterExpression` especifica que `#p` (`Price`) debe ser mayor que `:val` (150). La llamada a `PrintItem` hace referencia a una función personalizada, tal como se describe en [Imprimir un elemento](#).

Obtención de varios elementos utilizando expresiones y otros atributos del elemento

En el siguiente ejemplo se muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` y un conjunto de expresiones para obtener y luego imprimir todos los elementos que tengan un elemento `ProductCategory` `Bike`. Solo se devuelven los siguientes atributos del elemento: `Id`, `Title` y todos los atributos de `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("====");  
}
```

En el ejemplo anterior, la propiedad `ProjectionExpression` especifica los atributos que se deben devolver. La propiedad `ExpressionAttributeNames` especifica el marcador de posición `#pr` para que represente el atributo `ProductReviews` y el marcador de posición `#pc` para que represente el atributo `ProductCategory`. La propiedad `ExpressionAttributeValues` especifica el marcador de posición `:catg` para que represente el valor `Bike`. La propiedad `FilterExpression` especifica que `#pc` (`ProductCategory`) debe ser igual que `:catg` (`Bike`). La llamada a `PrintItem` hace referencia a una función personalizada, tal como se describe en [Imprimir un elemento](#).

Imprimir un elemento

El siguiente ejemplo muestra cómo imprimir los atributos y valores de un elemento. Este ejemplo se utiliza en los ejemplos anteriores que muestran cómo [Obtener un elemento individual utilizando expresiones y la clave principal del elemento](#), [Obtener varios elementos utilizando expresiones y la clave principal de la tabla](#) y [Obtener varios elementos utilizando expresiones y otros atributos del elemento](#).

```
// 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");
}

```

En el ejemplo anterior, cada valor de atributo tiene varias data-type-specific propiedades que se pueden evaluar para determinar el formato correcto para imprimir el atributo. Estas propiedades incluyen B, BOOL, BS, L, M, N, NS, NULL, S y SS, que corresponden a aquellas propiedades con el [formato de datos JSON](#). Para propiedades como B, N, NULL y S, si la propiedad correspondiente no es null, entonces el atributo será del tipo de datos no null correspondiente. En el caso de propiedades como BS LM,NS,, ySS, si Count es mayor que cero, el atributo es del tipo de non-zero-value datos correspondiente. Si todas las data-type-specific propiedades del atributo son cero null o son Count iguales a cero, el atributo corresponde al tipo de BOOL datos.

Crear o reemplazar un elemento utilizando expresiones

En el siguiente ejemplo se muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` y un conjunto de expresiones para actualizar el elemento que tenga un `Title 18-Bicycle 301`. Si el elemento no existe todavía, se añade uno nuevo.

```

// 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);

```

En el ejemplo anterior, la propiedad `ExpressionAttributeNames` especifica el marcador de posición `#title` para que represente el atributo `Title`. La propiedad `ExpressionAttributeValues` especifica el marcador de posición `:product` para que represente el valor `18-Bicycle 301`. La propiedad `ConditionExpression` especifica que `#title (Title)` debe ser igual que `:product (18-Bicycle 301)`. La llamada a `CreateItemData` hace referencia a la siguiente función 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;
}

```

En el ejemplo anterior, se devuelve un elemento de ejemplo con datos de muestra al intermediario. Se crean una serie de atributos y valores correspondientes utilizando tipos de datos como BOOL, L, M, N, NS, S y SS, que equivalen a los del [formato de datos JSON](#).

Actualizar un elemento utilizando expresiones

El siguiente ejemplo muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` y un conjunto de expresiones para cambiar el Title a 18" Girl's Bike para el elemento con un Id 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);

```

En el ejemplo anterior, la propiedad `ExpressionAttributeNames` especifica el marcador de posición `#title` para que represente el atributo `Title`. La propiedad `ExpressionAttributeValues` especifica el marcador de posición `:newproduct` para que represente el valor `18" Girl's Bike`. La propiedad `UpdateExpression` especifica que se cambie `#title` (`Title`) por `:newproduct` (`18" Girl's Bike`).

Eliminación de un elemento utilizando expresiones

El siguiente ejemplo muestra el método

`Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` y un conjunto de expresiones para eliminar el elemento con un `Id` 301 pero solo si el `Title` del elemento es `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);
```

En el ejemplo anterior, la propiedad `ExpressionAttributeNames` especifica el marcador de posición `#title` para que represente el atributo `Title`. La propiedad `ExpressionAttributeValues` especifica el marcador de posición `:product` para que represente el valor `18-Bicycle 301`. La propiedad `ConditionExpression` especifica que `#title (Title)` debe ser igual que `:product (18-Bicycle 301)`.

Más información

Para obtener más información y ejemplos de código, consulte:

- [Serie de DynamoDB: expresiones](#)
- [Acceso a atributos de elemento con expresiones de proyección](#)
- [Uso de marcadores de posición para nombres y valores de atributo](#)
- [Especificación de condiciones con expresiones de condición](#)
- [Modificación de elementos y atributos con expresiones de actualización](#)
- [Trabajar con elementos mediante la API AWS SDK para .NET de bajo nivel](#)
- [Consulta de tablas mediante la API de bajo nivel AWS SDK para .NET](#)
- [Escaneo de tablas mediante la API de bajo nivel AWS SDK para .NET](#)
- [Trabajar con índices secundarios locales mediante la API de bajo nivel AWS SDK para .NET](#)
- [Trabajar con índices secundarios globales mediante la API de bajo nivel AWS SDK para .NET](#)

Compatibilidad con JSON en Amazon DynamoDB

Note

La información de este tema es específica para proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

AWS SDK para .NET admite datos JSON cuando se trabaja con Amazon DynamoDB. Esto permite obtener datos con formato JSON de forma más sencilla a partir de tablas de DynamoDB e insertar documentos JSON en ellas.

Temas

- [Obtención de datos de una tabla de DynamoDB en formato JSON](#)
- [Inserción de datos con formato JSON en una tabla de DynamoDB](#)
- [Conversiones de tipos de datos de DynamoDB en JSON](#)
- [Más información](#)

Obtención de datos de una tabla de DynamoDB en formato JSON

En el siguiente ejemplo se muestra cómo obtener datos de una tabla de DynamoDB en 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
// }
```

En el ejemplo anterior, el método `ToJson` de la clase `Document` convierte un elemento de la tabla en una cadena con formato JSON. El elemento se recupera a través del método `GetItem` de la clase `Table`. Para determinar el elemento que se va a obtener, en este ejemplo, el `GetItem` método utiliza la clave hash-and-range principal del elemento de destino. Para determinar la tabla de la que se va a obtener el elemento, el método `LoadTable` de la clase `Table` usa una instancia de la clase `AmazonDynamoDBClient` y el nombre de la tabla de destino en DynamoDB.

Inserción de datos con formato JSON en una tabla de DynamoDB

En el siguiente ejemplo se muestra cómo usar formato JSON para insertar un elemento en una tabla de 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);
```

En el ejemplo anterior, el método `FromJson` de la clase `Document` convierte una cadena con formato JSON en un elemento. El elemento se inserta en la tabla a través del método `PutItem` de la clase `Table`, que usa la instancia de la clase `Document` que contiene el elemento. Para determinar la tabla en la que se va a insertar el elemento, se llama al método `LoadTable` de la clase `Table`, especificando una instancia de la clase `AmazonDynamoDBClient` y el nombre de la tabla de destino en DynamoDB.

Conversiones de tipos de datos de DynamoDB en JSON

Si llama al método `ToJson` de la clase `Document` y, después, en los datos JSON resultantes, llama al método `FromJson` para convertir de nuevo los datos JSON en una instancia de una clase `Document`, algunos tipos de datos de DynamoDB no se convertirán del modo esperado. En concreto:

- Los conjuntos de DynamoDB (los tipos `SS`, `NS` y `BS`) se convertirán en matrices JSON.
- Los conjuntos y escalares binarios de DynamoDB (los tipos `B` y `BS`) se convertirán en cadenas o listas de cadenas JSON codificadas en base64.

En este escenario, debe llamar al método `DecodeBase64Attributes` de la clase `Document` para reemplazar los datos JSON codificados en base64 con la representación binaria correcta. En el siguiente ejemplo se reemplaza un atributo de elemento escalar binario codificado en base64 en una instancia de una clase `Document`, denominada `Picture`, con la representación binaria correcta. En este ejemplo también se hace lo mismo para un atributo de elemento del conjunto binario codificado en base64 en la misma instancia de la clase `Document`, denominada `RelatedPictures`:

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Más información

Para obtener más información y ejemplos de programación de JSON con DynamoDB con AWS SDK para .NET, consulte:

- [Compatibilidad de JSON de DynamoDB](#)
- [Actualización de Amazon DynamoDB: JSON, capa gratuita ampliada, escalado flexible y elementos más grandes](#)

Uso de Amazon EC2

AWS SDK para .NET Es compatible con [Amazon EC2](#), que es un servicio web que proporciona una capacidad informática de tamaño variable. Esta capacidad informática sirve para crear y alojar sus sistemas de software.

APIs

AWS SDK para .NET Proporciona API para los clientes de Amazon EC2. Estas API permiten trabajar con características de EC2, como grupos de seguridad y pares de claves. También permiten controlar instancias de Amazon EC2. Esta sección contiene una pequeña cantidad de ejemplos que muestran los patrones que puede seguir al trabajar con ellos APIs. Para ver el conjunto completo de ellos APIs, consulta la [referencia de la AWS SDK para .NET API](#) (y desplázate hasta «Amazon.ec2").

Los Amazon EC2 APIs se proporcionan en el paquete [AWSSDK NuGet .EC2](#).

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el](#) proyecto. Revise también la información disponible en [Uso del SDK](#).

Acerca de los ejemplos

En los ejemplos de esta sección se muestra cómo usar clientes de Amazon EC2 y cómo administrar instancias de Amazon EC2.

En el [tutorial de instancias de spot de EC2](#) se muestra cómo solicitar instancias de spot de Amazon EC2. Las instancias de spot permiten acceder a la capacidad de EC2 sin utilizar por un precio inferior al precio bajo demanda.

Temas

- [Uso de grupos de seguridad en Amazon EC2](#)
- [Uso de pares de claves de Amazon EC2](#)
- [Visualización de regiones y zonas de disponibilidad de Amazon EC2](#)
- [Uso de instancias de Amazon EC2](#)
- [Tutorial de instancias de spot de Amazon EC2](#)

Uso de grupos de seguridad en Amazon EC2

En Amazon EC2, un grupo de seguridad funciona como un firewall virtual que controla el tráfico de red de una o varias instancias EC2. De forma predeterminada, EC2 asocia las instancias con un grupo de seguridad que no permite el tráfico entrante. Puede crear un grupo de seguridad que permita a sus instancias EC2 aceptar un tráfico determinado. Por ejemplo, si necesita conectarse a una instancia EC2 Windows, debe configurar el grupo de seguridad para permitir el tráfico RDP.

Para obtener más información sobre los grupos de seguridad, consulte [Grupos de seguridad de Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#).

Warning

EC2-Classic se retirará el 15 de agosto de 2022. Le recomendamos que migre de EC2-Classic a una VPC. Para obtener más información, consulte la entrada del blog [EC2-Classic Networking is Retiring – Here's How to Prepare](#).

Para obtener información sobre los requisitos APIs y requisitos previos, consulte la sección principal ([Uso de Amazon EC2](#)).

Temas

- [Enumeración de grupos de seguridad](#)
- [Creación de grupos de seguridad](#)
- [Actualización de grupos de seguridad](#)

Enumeración de grupos de seguridad

En este ejemplo, se muestra cómo utilizar el para AWS SDK para .NET enumerar los grupos de seguridad. Si proporciona un ID de [Amazon Virtual Private Cloud](#), la aplicación enumera los grupos de seguridad de esa VPC concreta. De lo contrario, la aplicación simplemente muestra una lista de todos los grupos de seguridad disponibles.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Enumeración de grupos de seguridad](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Enumeración de grupos de seguridad

El siguiente fragmento de código enumera los grupos de seguridad. Enumera todos los grupos, o bien los grupos de una VPC concreta, si se indica alguna.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [DescribeSecurityGroupsRequest](#)

Clase [DescribeSecurityGroupsResponse](#)

Clase [Filter](#)

Clase [SecurityGroup](#)

El 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();
    }
}
}
```

Consideraciones adicionales

- Observe que, en el caso de la VPC, el filtro se crea con la parte Name del par nombre-valor establecida en “vpc-id”. Este nombre proviene de la descripción de la [Filters](#) propiedad de la [DescribeSecurityGroupsRequest](#) clase.

- Para obtener la lista completa de sus grupos de seguridad, también puede [DescribeSecurityGroupsAsync](#) utilizarlos sin parámetros.
- Para comprobar los resultados, puede consultar la lista de grupos de seguridad en la [consola de Amazon EC2](#).

Creación de grupos de seguridad

En este ejemplo, se muestra cómo utilizar el AWS SDK para .NET para crear un grupo de seguridad. Puede proporcionar el ID de una VPC existente para crear un grupo de seguridad para EC2 en una VPC. Si no proporciona dicho identificador, el nuevo grupo de seguridad será para EC2-Classical si su AWS cuenta lo admite.

Si no proporciona un ID de VPC y su AWS cuenta no es compatible con EC2-Classical, el nuevo grupo de seguridad pertenecerá a la VPC predeterminada de su cuenta.

Warning

EC2-Classical se retirará el 15 de agosto de 2022. Le recomendamos que migre de EC2-Classical a una VPC. Para obtener más información, consulte la entrada del blog [EC2-Classical Networking se retira: cómo prepararse](#).

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Búsqueda de grupos de seguridad existentes](#)
- [Creación de un grupo de seguridad](#)
- [Código completo](#)

Búsqueda de grupos de seguridad existentes

El siguiente fragmento de código busca los grupos de seguridad existentes en la VPC indicada con el nombre especificado.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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;
}
```

Creación de un grupo de seguridad

El siguiente fragmento de código crea un grupo de seguridad nuevo si no existe uno con ese nombre en la VPC indicada. Si no se proporciona ninguna VPC y hay uno o más grupos con ese nombre, el fragmento de código simplemente devuelve la lista de grupos.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)
[EC2Cliente Class Amazon](#)
- Espacio de nombres [Amazon.EC2.Model](#)

Clase [CreateSecurityGroupRequest](#)

Clase [CreateSecurityGroupResponse](#)

Clase [DescribeSecurityGroupsRequest](#)

Clase [DescribeSecurityGroupsResponse](#)

Clase [Filter](#)

Clase [SecurityGroup](#)

El 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($"'\nNot 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($"'\nGroupName: {group.GroupName}");
    Console.WriteLine($"'\nGroupId: {group.GroupId}");
    Console.WriteLine($"'\nDescription: {group.Description}");
    Console.WriteLine($"'\nVpcId (if any): {group.VpcId}");
}
}

//
// Method to create a new security group (either EC2-Classic or EC2-VPC)
// If vpcID is empty, the security group will be for EC2-Classic
private static async Task<List<SecurityGroup>> CreateSecurityGroup(
    IAmazonEC2 ec2Client, string groupName, string vpcID)
{
    // See if one or more security groups with that name
    // already exist in the given VPC. If so, return the list of them.
    var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
    if (securityGroups.Count > 0)
    {
        Console.WriteLine(
            $"'\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
    }

    // If the security group doesn't already 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);  
}  
}
```

Actualización de grupos de seguridad

En este ejemplo, se muestra cómo utilizar la AWS SDK para .NET para añadir una regla a un grupo de seguridad. En concreto, el ejemplo agrega una regla para permitir el tráfico entrante en un determinado puerto TCP, que se puede usar, por ejemplo, en las conexiones remotas a una instancia de EC2. La aplicación toma el ID de un grupo de seguridad existente, una dirección IP (o un intervalo de direcciones) en formato CIDR y, opcionalmente, un número de puerto TCP. A continuación, agrega una regla de entrada al grupo de seguridad en cuestión.

Note

Para usar este ejemplo, necesita una dirección IP (o un intervalo de direcciones) en formato CIDR. Consulte la sección Consideraciones adicionales al final de este tema para conocer los métodos con los que obtener la dirección IP del equipo local.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Adición de una regla de entrada](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Adición de una regla de entrada

El siguiente fragmento de código agrega una regla de entrada a un grupo de seguridad para una dirección IP (o intervalo de direcciones) y un puerto TCP determinados.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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($"\\nNew RDP rule was written in {groupID} for {ipAddress}.");  
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");  
}
```

Código completo

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [AuthorizeSecurityGroupIngressRequest](#)

Clase [AuthorizeSecurityGroupIngressResponse](#)

Clase [IpPermission](#)

Clase [IpRange](#)

El 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);
}
}
}

```

Consideraciones adicionales

- Si no se indica un número de puerto, la aplicación lo establece de forma predeterminada en 3389, que es el puerto de Windows RDP, con el que puede conectarse a una instancia de EC2 donde se ejecute Windows. En cambio, si lanza una instancia de EC2 que ejecuta Linux, utilice el puerto TCP 22 (SSH).
- Observe que, en el ejemplo, `IpProtocol` está establecido en "tcp". Los valores de `IpProtocol` se encuentran en la descripción de la `IpProtocol` propiedad de la [IpPermission](#) clase.
- Es conveniente que tenga la dirección IP de su equipo local cuando utilice este ejemplo. Estas son algunas formas de obtener la dirección.
 - Si el equipo local (desde el que se conectará a la instancia de EC2) tiene una dirección IP pública estática, puede utilizar un servicio para obtener esa dirección. Uno de estos servicios es <http://checkip.amazonaws.com/>. Para obtener más información sobre la autorización del tráfico entrante, consulte [Agregar reglas a un grupo de seguridad](#) y [Reglas de grupos de seguridad para diferentes casos de uso](#) en la [Guía del usuario de Amazon EC2](#).
 - Otra forma de obtener la dirección IP del equipo local consiste en utilizar la [consola de Amazon EC2](#).

Seleccione uno de sus grupos de seguridad, seleccione la pestaña Reglas de entrada y, luego, seleccione Editar reglas de entrada. En una regla de entrada, abra el menú desplegable de la columna Origen y seleccione Mi IP para ver la dirección IP de su equipo local en formato CIDR. Asegúrese de cancelar la operación.

- Para comprobar los resultados de este ejemplo, puede examinar la lista de grupos de seguridad en la [consola de Amazon EC2](#).

Uso de pares de claves de Amazon EC2

Amazon EC2 utiliza la criptografía de clave pública para cifrar y descifrar la información de inicio de sesión. En la criptografía de clave pública se utiliza una clave pública para cifrar datos y, a continuación, el destinatario utiliza la clave privada para descifrar los datos. El conjunto de clave pública y clave privada se denomina par de claves. Si desea iniciar sesión en una instancia de EC2, debe especificar un par de claves al lanzar la instancia y, a continuación, proporcionar la clave privada del par cuando se conecte a ella.

Al lanzar una instancia de EC2, puede crear un par de claves exclusivo para ella o utilizar uno que ya haya utilizado al iniciar otras instancias. Para obtener más información sobre los pares de claves de Amazon EC2, consulte [Uso de pares de claves de Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#).

Para obtener información sobre los requisitos APIs y requisitos previos, consulte la sección principal ([Uso de Amazon EC2](#)).

Temas

- [Creación y visualización de pares de claves](#)
- [Eliminación de pares de claves](#)

Creación y visualización de pares de claves

En este ejemplo, se muestra cómo utilizar el AWS SDK para .NET para crear un key pair. La aplicación toma el nombre del nuevo par de claves y el nombre de un archivo PEM (que tiene una extensión “.pem”). Crea el par de claves, escribe la clave privada en el archivo PEM y, a continuación, muestra todos los pares de claves disponibles. Si no proporciona argumentos de línea de comandos, la aplicación simplemente muestra todos los pares de claves disponibles.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Creación del par de claves](#)
- [Visualización de los pares de claves disponibles](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Creación del par de claves

El siguiente fragmento de código crea un par de claves y, a continuación, almacena la clave privada del archivo PEM especificado.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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($"Created new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
        writer.WriteLine(response.KeyPair.KeyMaterial);
    }
}
```

Visualización de los pares de claves disponibles

En el siguiente fragmento de código se muestra una lista de los pares de claves disponibles.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [CreateKeyPairRequest](#)

Clase [CreateKeyPairResponse](#)

Clase [DescribeKeyPairsResponse](#)

Clase [KeyPairInfo](#)

El 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($"Created new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {

```

```

        writer.WriteLine(response.KeyPair.KeyMaterial);
    }
}

//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
}
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.

```



```

//
// If a key is found without a matching value, Dictionary.Value is set to the key
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

            // Check to see if there's a value that goes with this option?
            if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
            parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
            parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
            n++;
        }
    }

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(

```

```
Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
```

Consideraciones adicionales

- Tras ejecutar el ejemplo, puede ver el nuevo par de claves en la [consola de Amazon EC2](#).
- Al crear un par de claves, debe guardar la clave privada que se devuelve, ya que no podrá recuperarla más adelante.

Eliminación de pares de claves

En este ejemplo, se muestra cómo utilizar el AWS SDK para .NET para eliminar un key pair. La aplicación toma el nombre de una pareja de claves. Elimina el par de claves y, a continuación, muestra todos los pares de claves disponibles. Si no proporciona argumentos de línea de comandos, la aplicación simplemente muestra todos los pares de claves disponibles.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Eliminación del par de claves](#)
- [Visualización de los pares de claves disponibles](#)

- [Código completo](#)

Eliminación del par de claves

El siguiente fragmento de código elimina un par de claves.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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($"\\nKey pair {keyName} has been deleted (if it existed).");  
}
```

Visualización de los pares de claves disponibles

En el siguiente fragmento de código se muestra una lista de los pares de claves disponibles.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [DeleteKeyPairRequest](#)

Clase [DescribeKeyPairsResponse](#)

Clase [KeyPairInfo](#)

El 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}");
    }
}
}

```

Visualización de regiones y zonas de disponibilidad de Amazon EC2

Amazon EC2 está alojado en varias ubicaciones de todo el mundo. Dichas ubicaciones se componen de regiones y zonas de disponibilidad. Cada región es un área geográfica distinta y tiene varias ubicaciones aisladas, denominadas zonas de disponibilidad.

Para obtener más información acerca de las regiones y zonas de disponibilidad, consulte [Regiones y zonas](#) en la [Guía del usuario de Amazon EC2](#).

En este ejemplo, se muestra cómo utilizar el AWS SDK para .NET para obtener detalles sobre las regiones y las zonas de disponibilidad relacionadas con un cliente EC2. La aplicación muestra listas de las regiones y las zonas de disponibilidad disponibles para un cliente de EC2.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [DescribeAvailabilityZonesResponse](#)

Clase [DescribeRegionsResponse](#)

Clase [AvailabilityZone](#)

Clase [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);
}
}
}

```

Uso de instancias de Amazon EC2

Puede usarlo AWS SDK para .NET para controlar las instancias de Amazon EC2 con operaciones como crear, iniciar y terminar. En los temas de esta sección se proporcionan algunos ejemplos de cómo hacerlo. Para obtener más información sobre las instancias de EC2, consulte [Instancias de Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#).

Para obtener información sobre los requisitos APIs y requisitos previos, consulte la sección principal ([Uso de Amazon EC2](#)).

Temas

- [Lanzamiento de una instancia de Amazon EC2](#)
- [Terminación de una instancia de Amazon EC2](#)

Lanzamiento de una instancia de Amazon EC2

En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para lanzar una o más instancias de Amazon EC2 configuradas de forma idéntica desde la misma imagen de máquina de Amazon (AMI). La aplicación utiliza las [diferentes entradas](#) que proporcione para lanzar una instancia de EC2 y, a continuación, la monitoriza hasta que deja de estar en estado “Pendiente”.

Cuando la instancia de EC2 está en ejecución, puede conectarse a ella de forma remota, como se describe en [\(opcional\) Conexión a la instancia](#).

Warning

EC2-Classic se retirará el 15 de agosto de 2022. Le recomendamos que migre de EC2-Classic a una VPC. Para obtener más información, consulte la entrada del blog [EC2-Classic Networking se retira: cómo prepararse](#).

En las siguientes secciones se proporcionan fragmentos de código y otra información de este ejemplo. Después de los fragmentos de código se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Reunir todo lo necesario](#)
- [Iniciar una instancia](#)
- [Monitorización de la instancia](#)
- [Código completo](#)
- [Consideraciones adicionales](#)
- [\(opcional\) Conexión a la instancia](#)
- [Limpieza](#)

Reunir todo lo necesario

Para lanzar una instancia de EC2, necesitará varios elementos.

- Una [VPC](#) donde se lanzará la instancia. Si va a ser una instancia de Windows y se va a conectar a ella a través de RDP, lo más probable es que la VPC necesite tener asociada una puerta de enlace de Internet, así como una entrada correspondiente a esa puerta de enlace de Internet en la tabla de enrutamiento. Para obtener más información, consulte [Puertas de enlace de Internet](#) en la Guía del usuario de Amazon VPC.
- El ID de una subred existente en la VPC donde se lanzará la instancia. Una forma sencilla de encontrarlo o crearlo es iniciar sesión en la [consola de Amazon VPC](#), pero también puede obtenerlo mediante programación mediante los métodos y [CreateSubnetAsyncDescribeSubnetsAsync](#)

 Note

Si no proporciona este parámetro, la nueva instancia se lanza en la VPC predeterminada de la cuenta.

- El ID de un grupo de seguridad existente que pertenezca a la VPC donde se lanzará la instancia. Para obtener más información, consulte [Uso de grupos de seguridad en Amazon EC2](#).
- Si desea conectarse a la instancia nueva, el grupo de seguridad mencionado anteriormente debe tener una regla de entrada adecuada que permita el tráfico SSH en el puerto 22 (instancia de Linux) o el tráfico RDP en el puerto 3389 (instancia de Windows). Para obtener información sobre cómo hacerlo, consulte [Actualización de grupos de seguridad](#), así como la sección [Consideraciones adicionales](#) al final de ese tema.
- La imagen de máquina de Amazon (AMI) que se utilizará para crear la instancia. Para obtener más información AMIs, consulte [Amazon Machine Images \(AMIs\)](#) en la Guía del [usuario de Amazon EC2](#). En concreto, consulte [Buscar una AMI](#) y [compartirla AMIs](#).
- El nombre de un par de claves de EC2 existente, que se utiliza para conectarse a la nueva instancia. Para obtener más información, consulte [Uso de pares de claves de Amazon EC2](#).
- El nombre del archivo PEM que contiene la clave privada del par de claves de EC2 mencionado anteriormente. El archivo PEM se utiliza cuando se [conecta de forma remota](#) a la instancia.

Iniciar una instancia

El siguiente fragmento de código lanza una instancia de EC2.

El ejemplo que aparece [cerca del final de este tema](#) muestra este fragmento de código en 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;
}
```

Monitorización de la instancia

El siguiente fragmento de código monitoriza la instancia hasta que deja de estar en estado “Pendiente”.

El ejemplo que aparece [cerca del final de este tema](#) muestra este fragmento de código en uso.

Consulte la [InstanceState](#) clase para ver los valores válidos de la `Instance.State.Code` propiedad.

```
//
// 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}");
}

```

```
}
```

Código completo

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

Clase [InstanceType](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [DescribeInstancesRequest](#)

Clase [DescribeInstancesResponse](#)

Clase [Instance](#)

Clase [InstanceNetworkInterfaceSpecification](#)

Clase [RunInstancesRequest](#)

Clase [RunInstancesResponse](#)

El 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.Write(".");

```

```

        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach (Instance i in responseDescribe.Reservations[0].Instances)
        {
            // Check the lower byte of State.Code property
            // Code == 0 is the pending state
            if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
            break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
            break;
    }

    Console.WriteLine("\nNo more instances are pending.");
    foreach (Instance i in responseDescribe.Reservations[0].Instances)
    {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
    }
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s  

<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
}
}

```



```

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];
                var value = key;

                // Check to see if there's a value that goes with this option?
                if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
                parsedArgs.Add(key, value);
            }

            // If the first argument in this iteration doesn't start with a dash, it's a
value
            else
            {
                parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
                n++;
            }
        }
    }
}

```

```

    }
}

return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}
}
}
}

```

Consideraciones adicionales

- Al comprobar el estado de una instancia EC2, puede añadir un filtro a la `Filter` propiedad del [DescribeInstancesRequest](#) objeto. Con esta técnica, puede limitar la solicitud a unas instancias determinadas, por ejemplo, las instancias con una etiqueta concreta especificada por el usuario.

- En aras de una mayor brevedad, se han asignado valores típicos a algunas propiedades, pero cualquiera o todas estas propiedades se pueden determinar mediante programación o usando entradas de usuario.
- Los valores que puede utilizar para el [RunInstancesRequest](#) objeto MinCount y MaxCount sus propiedades vienen determinados por la zona de disponibilidad de destino y el número máximo de instancias permitido para el tipo de instancia. Para obtener más información, consulte [¿Cuántas instancias puedo ejecutar en Amazon EC2?](#) en las preguntas frecuentes generales de Amazon EC2.
- Si desea utilizar un tipo de instancia diferente al de este ejemplo, puede elegir entre varios tipos. Para obtener más información, consulte [Tipos de instancias de Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#). Consulte también [Detalles del tipo de instancia](#) y [Explorador de tipos de instancia](#).
- También puede asociar un [rol de IAM](#) a una instancia al lanzarla. Para ello, cree un [IamInstanceProfileSpecification](#) objeto cuya Name propiedad esté establecida en el nombre de un rol de IAM. A continuación, añada ese objeto a la IamInstanceProfile propiedad del [RunInstancesRequest](#) objeto.

 Note

Para lanzar una instancia de EC2 que tenga un rol de IAM asociado, la configuración de un usuario de IAM debe incluir determinados permisos. Para obtener más información sobre los permisos necesarios, consulte la sección [Concesión de permisos a un usuario para transferir un rol de IAM a una instancia](#) en la [Guía del usuario de Amazon EC2](#).

(opcional) Conexión a la instancia

En cuanto se esté ejecutando una instancia, puede conectarse de forma remota a ella utilizando el cliente remoto adecuado. Tanto si las instancias son de Linux o de Windows, necesita la dirección IP pública o el nombre DNS público de la instancia. También necesitará lo siguiente.

Instancias de Linux

Puede usar un cliente SSH para conectarse a la instancia de Linux. Asegúrese de que el grupo de seguridad que utilizó al lanzar la instancia permite el tráfico SSH en el puerto 22, tal y como se describe en [Actualización de grupos de seguridad](#).

También necesitará el archivo que contiene la parte privada de par de claves que usó para lanzar la instancia, es decir, el archivo PEM.

Para obtener más información, consulte [Conexión con la instancia de Linux](#) en la Guía del usuario de Amazon EC2.

Instancias de Windows

Puede usar un cliente RDP para conectarse a la instancia. Asegúrese de que el grupo de seguridad que utilizó al lanzar la instancia permite el tráfico RDP en el puerto 3389, tal y como se describe en [Actualización de grupos de seguridad](#).

También necesitará una contraseña de administrador. Puede obtenerlo mediante el siguiente código de ejemplo, que requiere el ID de la instancia y la parte privada del par de claves que usó para lanzar la instancia, es decir, el archivo PEM.

Para obtener más información, consulte [Conexión a la instancia de Windows](#) en la Guía del usuario de Amazon EC2.

Warning

Este código de ejemplo devuelve la contraseña de administrador de la instancia en texto si formato.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [GetPasswordDataRequest](#)

Clase [GetPasswordDataResponse](#)

El 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);
}
}
}
}

```

Limpieza

Cuando ya no necesite la instancia de EC2, asegúrese de terminarla, como se describe en [Terminación de una instancia de Amazon EC2](#).

Terminación de una instancia de Amazon EC2

Cuando ya no necesite una o varias de las instancias de Amazon EC2, puede terminarlas.

En este ejemplo, se muestra cómo utilizarlos AWS SDK para .NET para terminar las instancias de EC2. Se toma como entrada un ID de instancia.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [TerminateInstancesRequest](#)

Clase [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);
    }
}
}
}
}

```

Tras ejecutar el ejemplo, es conveniente iniciar sesión en la [consola de Amazon EC2](#) para confirmar que la [instancia EC2](#) se ha terminado.

Tutorial de instancias de spot de Amazon EC2

En este tutorial, se muestra cómo utilizarlos AWS SDK para .NET para gestionar las instancias puntuales de Amazon EC2.

Descripción general de

Las instancias de spot permiten solicitar capacidad de Amazon EC2 sin utilizar por un precio inferior al precio bajo demanda. Esto puede reducir considerablemente los costos de EC2 correspondientes a aplicaciones que pueden interrumpirse.

Este es un resumen general de cómo solicitar y utilizar instancias de spot.

1. Cree una solicitud de instancia de spot, especificando el precio máximo que está dispuesto a pagar.
2. Una vez atendida la solicitud, ejecute la instancia como lo haría con cualquier otra instancia de Amazon EC2.
3. Ejecute la instancia todo el tiempo que desee y, a continuación, téminela, a menos que el precio de instancia spot cambie y la instancia termine automáticamente.
4. Limpie la solicitud de instancia de spot cuando ya no la necesite para que no se creen más instancias de spot.

Esta descripción general de las instancias de spot es muy genérica. Para comprender mejor las instancias de spot, consulte [Instancias de spot](#) en la [Guía del usuario de Amazon EC2](#).

Acerca de este tutorial

Al seguir este tutorial, utilizará el AWS SDK para .NET para hacer lo siguiente:

- Creación de una solicitud de instancia de spot
- Determinar cuándo se ha atendido la solicitud de instancia de spot
- Cancelar la solicitud de instancia de spot
- Terminar las instancias asociadas

En las siguientes secciones se proporcionan fragmentos de código y otra información de este ejemplo. Después de los fragmentos de código se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Requisitos previos](#)
- [Reunir todo lo necesario](#)
- [Creación de una solicitud de instancia de spot](#)
- [Determinación del estado de la solicitud de instancia de spot](#)
- [Limpieza de las solicitudes de instancia de spot](#)
- [Limpieza de las instancias de spot](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Requisitos previos

Para obtener información sobre los requisitos APIs y requisitos previos, consulte la sección principal ([Uso de Amazon EC2](#)).

Reunir todo lo necesario

Para crear una solicitud de instancia de spot, necesitará varias cosas.

- El número de instancias y el tipo de instancia. Hay varios tipos de instancias para elegir. Para obtener más información, consulte [Tipos de instancias de Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#). Consulte también [Detalles del tipo de instancia](#) y [Explorador de tipos de instancia](#).

En este tutorial, el número predeterminado es 1.

- La imagen de máquina de Amazon (AMI) que se utilizará para crear la instancia. Para obtener más información AMIs, consulte [Amazon Machine Images \(AMIs\)](#) en la Guía del [usuario de Amazon EC2](#). En concreto, consulte [Buscar una AMI](#) y [compartirla AMIs](#).
- El precio máximo que está dispuesto a pagar por hora de instancia. Puede ver los precios de todos los tipos de instancias (tanto de las instancias bajo demanda como de las instancias de spot) en la [página de precios de Amazon EC2](#). El precio predeterminado de este tutorial se explica más adelante.
- Si quiere conectarse a una instancia de forma remota, un grupo de seguridad con la configuración y los recursos adecuados. Esto se describe en [Uso de grupos de seguridad en Amazon EC2](#) y en la información sobre cómo [reunir todo lo necesario](#) y cómo [conectarse a una instancia](#) en [Lanzamiento de una instancia de Amazon EC2](#). Para que todo sea más sencillo, en este tutorial se utiliza el grupo de seguridad predeterminado que tienen todas las cuentas de AWS más nuevas.

Hay muchas formas de abordar la solicitud de instancias de spot. Estas son algunas estrategias comunes:

- Realizar solicitudes que seguro que van a costar menos que el precio bajo demanda
- Realizar solicitudes según en el valor del cálculo resultante
- Realizar solicitudes adquirir capacidad informática lo más rápidamente posible

Las siguientes explicaciones hacen referencia al [historial de precios de las instancias de spot](#) de la [Guía del usuario de Amazon EC2](#).

Reducir el costo por debajo del precio bajo demanda

Tiene una tarea de procesamiento por lotes que tardará en ejecutarse una cantidad determinada de horas o días. Sin embargo, es flexible con respecto a cuándo comienza y finaliza. Desea ver si puede completarla por menos del valor del costo de las instancias bajo demanda.

Examina el historial de precios de instancia de spot de tipos de instancia usando la consola de administración de Amazon EC2 o la API de Amazon EC2. Una vez que haya analizado el historial de precios para su tipo de instancia deseado en una zona de disponibilidad especificada, tendrá dos enfoques alternativos para su solicitud:

- Especificar una solicitud en el extremo superior del rango de precios de instancias de spot, que aún son inferiores al precio bajo demanda, contando con que lo más probable es que su solicitud de instancia de spot puntual se atienda y se ejecute durante un tiempo de computación consecutivo suficiente para completar la tarea.
- Especificar una solicitud en el extremo inferior del rango de precios y tener previsto combinar muchas instancias lanzadas con el tiempo a través de una solicitud persistente. Las instancias se ejecutarían el tiempo suficiente, combinadas, para completar la tarea con un costo total aún más reducido

No pagar un importe superior al valor del resultado

Tiene una tarea de procesamiento de datos para ejecutar. Conoce las ventajas de los resultados de la tarea lo suficientemente bien como para saber lo valiosos que son en términos de costos de computación.

Una vez que analizado el historial de precios de instancias de spot del tipo de instancia, elige un precio en el que el costo del tiempo de computación no es superior al valor de los resultados de la tarea. Crea una solicitud persistente y deja que se ejecute intermitentemente a medida que el precio de instancia de spot fluctúa en torno a su solicitud o por debajo de esta.

Adquirir capacidad de computación con rapidez

Tiene una necesidad a corto plazo no anticipada de capacidad adicional que no está disponible a través de las instancias bajo demanda. Una vez analizado el historial de precios de instancia de spot del tipo de instancia, elige un precio por encima del precio histórico para mejorar significativamente la probabilidad de que la solicitud se atienda con rapidez y continúe computándose hasta completarse.

Tras reunir todo lo necesario y elegir una estrategia, está listo para solicitar una instancia de spot. En este tutorial, el precio máximo de instancias de spot predeterminado se establece que sea el mismo que el precio bajo demanda (que es 0,003 USD para este tutorial). Establecer el precio de esta manera maximiza las posibilidades de que se cumpla la solicitud.

Creación de una solicitud de instancia de spot

En el siguiente fragmento de código se muestra cómo crear una solicitud de instancia de spot con los elementos que ha reunido anteriormente.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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];  
}
```

El valor importante que devuelve este método es el identificador de solicitud de la instancia puntual, que se encuentra en el `SpotInstanceRequestId` elemento del [SpotInstanceRequest](#) objeto devuelto.

 Note

Se le cobrará por cualquier instancia de spot que se lance. Para evitar costos innecesarios, asegúrese de [cancelar todas las solicitudes](#) y de [terminar cualquier instancia](#).

Determinación del estado de la solicitud de instancia de spot

En el siguiente fragmento de código se muestra cómo obtener información sobre la solicitud de instancia de spot. Puede utilizar esa información para tomar algunas decisiones relativas a su código, por ejemplo, si debe seguir esperando a que se atienda una solicitud de instancia de spot.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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];  
}
```

El método devuelve información sobre la solicitud de instancia de spot, como el ID de la instancia, su estado y el código de estado. Para obtener más información sobre los códigos de estado de las solicitudes de instancias de spot, consulte [Estado de las solicitudes de spot](#) en la [Guía del usuario de Amazon EC2](#).

Limpieza de las solicitudes de instancia de spot

Cuando ya no necesite solicitar instancias de spot, es importante cancelar las solicitudes pendientes para evitar que se vuelvan a atenderse. En el siguiente fragmento de código se muestra cómo cancelar una solicitud de instancia de spot.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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);
}
```

Limpieza de las instancias de spot

Para no incurrir en costos innecesarios, es importante terminar cualquier instancia que se haya iniciado a partir de solicitudes de instancia de spot; las instancias no terminarán si solamente se cancelan, lo que significa que se le seguirá cobrando por ellas. En el siguiente fragmento de código se muestra cómo terminar una instancia después de obtener el identificador de instancia de una instancia de spot activa.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En el siguiente ejemplo de código se utilizan los métodos descritos anteriormente para crear y cancelar una solicitud de instancia de spot y terminar una instancia de spot.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.EC2](#)

Elementos de programación:

- Espacio de nombres [Amazon.EC2](#)

[EC2Cliente Class Amazon](#)

Clase [InstanceType](#)

- Espacio de nombres [Amazon.EC2.Model](#)

Clase [CancelSpotInstanceRequestsRequest](#)

Clase [DescribeSpotInstanceRequestsRequest](#)

Clase [DescribeSpotInstanceRequestsResponse](#)

Clase [InstanceStateChange](#)

Clase [LaunchSpecification](#)

Clase [RequestSpotInstancesRequest](#)

Clase [RequestSpotInstancesResponse](#)

Clase [SpotInstanceRequest](#)

Clase [TerminateInstancesRequest](#)

Clase [TerminateInstancesResponse](#)

El 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");
        }
    }
}

```

```
}  
}  
}  
}  
}
```

Consideraciones adicionales

- Tras ejecutar el tutorial, se recomienda iniciar sesión en la [consola de Amazon EC2](#) para confirmar que la [solicitud de instancia de spot](#) se ha cancelado y que la [instancia de spot](#) se ha terminado.

Acceder AWS Identity and Access Management (IAM) con el AWS SDK para .NET

The AWS SDK para .NET supports [AWS Identity and Access Management](#), que es un servicio web que permite a AWS los clientes gestionar los usuarios y los permisos de los usuarios AWS.

Un usuario AWS Identity and Access Management (IAM) es una entidad en AWS la que se crea. La entidad representa a una persona o aplicación con la que interactúa. AWS Para obtener más información sobre los usuarios de IAM, consulte [Usuarios de IAM](#) e [IAM y cuotas de AWS STS](#) en la Guía del usuario de IAM.

Para conceder permisos a un usuario, hay que crear una política de IAM. Una política es un documento de política que incluye una lista de las acciones que un usuario puede realizar y los recursos a los que pueden afectar dichas acciones. Para obtener más información sobre las políticas de IAM, consulte [Políticas y permisos](#) en la Guía del usuario de IAM.

Warning

Para evitar riesgos de seguridad, no utilice a los usuarios de IAM para la autenticación cuando desarrolle software especialmente diseñado o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como [AWS IAM Identity Center](#).

APIs

Los proveedores son AWS SDK para .NET APIs para los clientes de IAM. Le APIs permiten trabajar con funciones de IAM, como usuarios, roles y claves de acceso.

Esta sección contiene una pequeña cantidad de ejemplos que muestran los patrones que puede seguir al trabajar con ellos APIs. Para ver el conjunto completo de APIs, consulta la [referencia de la AWS SDK para .NET API](#) (y desplázate hasta «Amazon. IdentityManagement»).

En esta sección también se incluye [un ejemplo](#) que muestra cómo asociar un rol de IAM a instancias de Amazon EC2 para administrar las credenciales más fácilmente.

[Los IAM APIs los proporciona elAWS SDK. IdentityManagement](#) NuGetpaquete.

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información en [Uso del SDK](#).

Temas

Temas

- [Creación de políticas administradas de IAM a partir de JSON](#)
- [Visualización del documento de política de una política administrada de IAM](#)
- [Concesión de acceso mediante un rol de IAM](#)

Creación de políticas administradas de IAM a partir de JSON

En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para crear una [política gestionada de IAM](#) a partir de un documento de política determinado en JSON. La aplicación crea un objeto de cliente de IAM, lee el documento de política de un archivo y, a continuación, crea la política.

Note

Para ver un ejemplo de documento de política en JSON, consulte la sección [Consideraciones adicionales](#) al final de este tema.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Creación de la política](#)

- [Código completo](#)
- [Consideraciones adicionales](#)

Creación de la política

El siguiente fragmento de código crea una política administrada de IAM con el nombre y el documento de política especificados.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.IdentityManagement](#)

Elementos de programación:

- [Espacio de nombres Amazon. IdentityManagement](#)

Clase [AmazonIdentityManagementServiceClient](#)

- [Espacio de nombres Amazon. IdentityManagement](#).Modelo

Clase [CreatePolicyRequest](#)

Clase [CreatePolicyResponse](#)

El 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);
}
}
}
```

Consideraciones adicionales

- El siguiente es un ejemplo de documento de política que se puede copiar en un archivo JSON y utilizar como entrada para esta aplicación:

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": "*"
}
]
```

- Para confirmar que la política se ha creado, consulte la [consola de IAM](#). En la lista desplegable Filtrar políticas, seleccione Administrado por el cliente. Elimine la política cuando ya no la necesite.
- Para obtener más información sobre la creación de políticas, consulte [Crear políticas de IAM](#) y la [Referencia de políticas JSON de IAM](#) en la [Guía del usuario de IAM](#).

Visualización del documento de política de una política administrada de IAM

En este ejemplo, se muestra cómo utilizar el AWS SDK para .NET para mostrar un documento de política. La aplicación crea un objeto de cliente de IAM, busca la versión predeterminada de la política administrada de IAM indicada y, a continuación, muestra el documento de política en JSON.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Búsqueda de la versión predeterminada](#)
- [Visualización del documento de política](#)
- [Código completo](#)

Búsqueda de la versión predeterminada

En el siguiente fragmento de código se busca la versión predeterminada de la política de IAM indicada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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;
}
```

Visualización del documento de política

El siguiente fragmento de código muestra el documento de política en JSON de la política de IAM indicada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.IdentityManagement](#)

Elementos de programación:

- [Espacio de nombres Amazon. IdentityManagement](#)
Clase [AmazonIdentityManagementServiceClient](#)
- [Espacio de nombres Amazon. IdentityManagement](#).Modelo

Clase [GetPolicyVersionRequest](#)

Clase [GetPolicyVersionResponse](#)

Clase [ListPolicyVersionsRequest](#)

Clase [ListPolicyVersionsResponse](#)

Clase [PolicyVersion](#)

El 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 responseVersions.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)}");
}
}
}

```

Concesión de acceso mediante un rol de IAM

En este tutorial, se muestra cómo utilizarlos AWS SDK para .NET para habilitar las funciones de IAM en las instancias de Amazon EC2.

Descripción general de

Todas las solicitudes AWS deben estar firmadas criptográficamente con las credenciales emitidas por AWS. Por tanto, se necesita una estrategia para administrar las credenciales de las aplicaciones que se ejecutan en instancias de Amazon EC2. Debe distribuir, almacenar y rotar estas credenciales de forma segura, pero también mantenerlas accesibles en las aplicaciones.

Con los roles de IAM, estas credenciales se pueden administrar de forma eficaz. Debe crear un rol de IAM y configurarlo con los permisos que una aplicación requiere y, a continuación, asociar ese rol a una instancia de EC2. Para obtener más información sobre las ventajas de usar roles de IAM, consulte [Roles de IAM para Amazon EC2](#) en la [Guía del usuario de Amazon EC2](#). Consulte también la información sobre [roles de IAM](#) de la Guía del usuario de IAM.

En el caso de una aplicación creada con el AWS SDK para .NET, cuando la aplicación crea un objeto de cliente para un AWS servicio, el objeto busca credenciales de varias fuentes posibles. En [Resolución de credencial y perfil](#) se muestra el orden de la búsqueda.

Si el objeto de cliente no encuentra credenciales en ningún origen, recupera unas credenciales temporales que tienen los mismos permisos que los asociados al rol de IAM y que están en los metadatos de la instancia de EC2. Estas credenciales se utilizan para realizar llamadas AWS desde el objeto cliente.

Acerca de este tutorial

Al seguir este tutorial, utilizará la AWS SDK para .NET (y otras herramientas) para lanzar una instancia de Amazon EC2 con una función de IAM asociada y, a continuación, verá una aplicación en la instancia con los permisos de la función de IAM.

Temas

- [Creación de una aplicación de Amazon S3 de ejemplo](#)
- [Creación de un rol de IAM](#)
- [Lanzamiento de una instancia de EC2 y asociación del rol de IAM](#)
- [Conexión a la instancia EC2](#)
- [Ejecutar la aplicación de muestra en la instancia EC2](#)
- [Limpieza](#)

Creación de una aplicación de Amazon S3 de ejemplo

Esta aplicación de ejemplo recupera un objeto de Amazon S3. Para ejecutar la aplicación de ejemplo, necesita lo siguiente:

- Un bucket de Amazon S3 que contiene un archivo de texto
- AWS credenciales de su máquina de desarrollo que le permiten acceder al depósito.

Para obtener información acerca de cómo crear un bucket de Amazon S3 y cómo cargar un archivo, consulte [Guía del usuario de Amazon Simple Storage Service](#). Para obtener información sobre AWS las credenciales, consulte [Autenticar el con AWS SDK para .NET AWS](#).

Cree un proyecto .NET Core con el siguiente código. A continuación, pruebe la aplicación en el equipo de desarrollo.

Note

En el equipo de desarrollo está instalado el tiempo de ejecución de .NET Core, que permite ejecutar la aplicación sin publicarla. Al crear una instancia de EC2 más adelante en este tutorial, puede optar por instalar el tiempo de ejecución de .NET Core en esa instancia. Esto le proporciona una experiencia similar, y la transferencia de archivos es más pequeña. Sin embargo, también puede optar por no instalar el tiempo de ejecución de .NET Core en la instancia. Si se decanta por esta opción, debe publicar la aplicación de forma que se incluya todas las dependencias al transferirla a la instancia.

Referencias de SDK

NuGet paquetes:

- [AWSSDKS.3](#)

Elementos de programación:

- Espacio de nombres [Amazon.S3](#)

Clase [AmazonS3Client](#)

- Espacio de nombres [Amazon.S3.Model](#)

Clase [GetObjectResponse](#)

El 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);
}
}
}
```

Si lo desea, puede eliminar temporalmente las credenciales que use en el equipo de desarrollo para ver cómo responde la aplicación (no se olvide de restaurarlas cuando acabe).

Creación de un rol de IAM

Cree un rol de IAM que tenga los permisos adecuados para obtener acceso a Amazon S3.

1. Abra la [consola de IAM](#).
2. En el panel de navegación, seleccione Roles y luego seleccione Crear rol.
3. Seleccione Servicio de AWS , busque y seleccione EC2 y seleccione Siguiente: Permisos.
4. En Adjuntar políticas de permisos, busca y selecciona AmazonS3 ReadOnlyAccess. Revise la política si lo desea y, a continuación, seleccione Siguiente: Etiquetas.
5. Agregue etiquetas si lo desea y, a continuación, seleccione Siguiente: Revisar.

6. Escriba un nombre y una descripción para el rol y, a continuación, elija Crear rol. Recuerde este nombre, ya que lo necesitará cuando lance su instancia EC2.

Lanzamiento de una instancia de EC2 y asociación del rol de IAM

Lance una instancia de EC2 con el rol de IAM que creó anteriormente. Puede hacerlo de las siguientes maneras:

- Con la consola de EC2

Para iniciar una instancia mediante la consola de EC2, consulte [Iniciación de una instancia con el nuevo asistente de inicialización de instancias](#) en la [Guía del usuario de Amazon EC2](#).

Al revisar la página de inicio, al menos deberá expandir el panel Detalles avanzados para que pueda especificar el rol de IAM que creó anteriormente en el perfil de instancia de IAM.

- Usando el AWS SDK para .NET

Para obtener información al respecto, consulte [Lanzamiento de una instancia de Amazon EC2](#), así como la sección [Consideraciones adicionales](#) al final de ese tema.

Para lanzar una instancia de EC2 que tenga un rol de IAM asociado, la configuración de un usuario de IAM debe incluir determinados permisos. Para obtener más información sobre los permisos necesarios, consulte la sección [Concesión de permisos a un usuario para transferir un rol de IAM a una instancia](#) en la [Guía del usuario de Amazon EC2](#).

Conexión a la instancia EC2

Conéctese a la instancia EC2 para poder transferirle la aplicación de muestra y luego ejecutarla. Necesitarás el archivo que contiene la parte privada del par de claves que usaste para lanzar la instancia, es decir, el archivo PEM.

Para obtener más información acerca de cómo conectarse a una instancia, consulte [Conexión con la instancia de Linux](#) o [Conexión con la instancia de Windows](#) en la [Guía del usuario de Amazon EC2](#). Cuando se conecte, hágalo de forma que pueda transferir archivos desde el equipo de desarrollo a la instancia.

Si usa Visual Studio en Windows, también se puede conectar a la instancia mediante el Kit de herramientas para Visual Studio. Para obtener más información, consulte [Conexión a una instancia de Amazon EC2](#) en la Guía del AWS Toolkit for Visual Studio usuario.

Ejecutar la aplicación de muestra en la instancia EC2

1. Copie los archivos de la aplicación de la unidad local en la instancia.

Los archivos que transfiera dependen de cómo se haya creado la aplicación y de si la instancia tiene instalado el tiempo de ejecución de .NET Core. Para obtener información sobre cómo transferir archivos a su instancia, consulte [Conexión con la instancia de Linux](#) (consulte la subsección correspondiente) o [Transferencia de archivos a instancias de Windows](#) en la [Guía del usuario de Amazon EC2](#).

2. Inicie la aplicación y compruebe que se ejecuta con los mismos resultados que en el equipo de desarrollo.
3. Confirme que la aplicación usa las credenciales proporcionadas por el rol de IAM.
 - a. Abra la [consola de Amazon EC2](#).
 - b. Seleccione la instancia y desasocie el rol de IAM mediante Acciones, Configuración de la instancia. Asociar o reemplazar rol de IAM.
 - c. Vuelva a ejecutar la aplicación y vea que devuelve un error de autorización.

Limpieza

Cuando acabe este tutorial, si ya no quiere conservar la instancia de EC2 que ha creado, asegúrese de terminarla para no incurrir en costos no deseados. Puede hacerlo en la [consola de Amazon EC2](#) o mediante programación, tal y como se describe en [Terminación de una instancia de Amazon EC2](#). Si lo desea, también puede eliminar otros recursos que haya creado durante este tutorial. Estos pueden englobar un rol de IAM, un par de claves de EC2 y un archivo PEM, un grupo de seguridad, etc.

Uso del servicio de almacenamiento de Internet de Amazon Simple Storage Service

AWS SDK para .NET Es compatible con [Amazon S3](#), que es almacenamiento para Internet. Está diseñado para facilitar a los desarrolladores recursos de computación escalables basados en Web.

APIs

Los proveedores AWS SDK para .NET son APIs para los clientes de Amazon S3. Le APIs permiten trabajar con los recursos de Amazon S3, como depósitos y elementos. Para ver el conjunto completo APIs de Amazon S3, consulte lo siguiente:

- [AWS SDK para .NET Referencia de API](#) (y desplázate hasta «Amazon.S3”).
- Documentación de [Amazon.Extensions.S3.Encryption](#)

Los Amazon S3 APIs vienen incluidos en los siguientes NuGet paquetes:

- [AWSSDKS.3](#)
- [Amazon.Extensions.S3.Encryption](#)

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información disponible en [Uso del SDK](#).

Ejemplos en este documento

Los siguientes temas de este documento muestran cómo utilizarlos AWS SDK para .NET para trabajar con Amazon S3.

- [Uso de claves de KMS para el cifrado de S3](#)

Ejemplos en otros documentos

Los siguientes enlaces a la [guía para desarrolladores de Amazon S3](#) proporcionan ejemplos adicionales de cómo utilizarla AWS SDK para .NET para trabajar con Amazon S3.

Note

Si bien estos ejemplos y otras consideraciones de programación se crearon para la versión 3 de la versión que AWS SDK para .NET utiliza .NET Framework, también son válidos para versiones posteriores de la versión que AWS SDK para .NET utiliza .NET Core. A veces es necesario realizar pequeños ajustes en el código.

Ejemplos de programación de Amazon S3

- [Administrar ACLs](#)
- [Creación de un bucket](#)
- [Carga de un objeto](#)
- [Carga de varias partes con la API de alto nivel \(Amazon.S3.Transfer\). TransferUtility\)](#)
- [Carga multiparte con la API de bajo nivel](#)
- [Listas de objetos](#)
- [Lista de claves](#)
- [Obtención de un objeto](#)
- [Copia de un objeto](#)
- [Copia de un objeto con la API de carga multiparte](#)
- [Eliminación de un objeto](#)
- [Eliminación de varios objetos](#)
- [Restauración de un objeto](#)
- [Configuración de un bucket para las notificaciones](#)
- [Administración del ciclo de vida de un objeto](#)
- [Generación de una URL de objeto prefirmada](#)
- [Administración de sitios web](#)
- [Habilitación del uso compartido de recursos entre orígenes \(CORS\)](#)

Consideraciones de programación adicionales

- [Uso de AWS SDK para .NET para la programación en Amazon S3](#)
- [Realización de solicitudes con las credenciales temporales de usuario de IAM](#)
- [Realización de solicitudes con credenciales temporales de usuario federado](#)
- [Especificación de cifrado del lado del servidor](#)
- [Especificación de cifrado del lado del servidor con claves de cifrado proporcionadas por el cliente](#)

Uso de AWS KMS claves para el cifrado de Amazon S3 en AWS SDK para .NET

En este ejemplo, se muestra cómo utilizar AWS Key Management Service las claves para cifrar objetos de Amazon S3. La aplicación crea una clave maestra del cliente (CMK) y la utiliza para crear

un objeto [EncryptionClientV2 de AmazonS3](#) para el cifrado del lado del cliente. La aplicación utiliza ese cliente para crear un objeto cifrado a partir de un determinado archivo de texto en un bucket de Amazon S3 existente. Luego, descifra el objeto y muestra su contenido.

 Warning

Una clase similar, `AmazonS3EncryptionClient`, está en desuso y es menos segura que la clase `AmazonS3EncryptionClientV2`. Para migrar código existente que use `AmazonS3EncryptionClient`, consulte [Migración del cliente de cifrado S3 \(V1 a V2\)](#).

Temas

- [Creación de materiales de cifrado](#)
- [Creación y cifrado de un objeto de Amazon S3](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Creación de materiales de cifrado

El siguiente fragmento de código crea un objeto `EncryptionMaterials` que contiene un ID de clave de KMS.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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);
```

Creación y cifrado de un objeto de Amazon S3

El siguiente fragmento de código crea un objeto `AmazonS3EncryptionClientV2` que utiliza los materiales de cifrado creados anteriormente. Luego, utiliza el cliente para crear y cifrar un nuevo objeto de Amazon S3.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [Amazon.Extensions.S3.Encryption](#)

Elementos de programación:

- Espacio de nombres [Amazon.Extensions.S3.Encryption](#)

Clase [Amazon S3 V2 EncryptionClient](#)

Clase [Amazon S3 V2 CryptoConfiguration](#)

Clase [CryptoStorageMode](#)

Clase [EncryptionMaterialsV2](#)

- Espacio de nombres [Amazon.Extensions.S3.Encryption.Primitives](#)

Clase [KmsType](#)

- Espacio de nombres [Amazon.S3.Model](#)

Clase [GetObjectRequest](#)

Clase [GetObjectResponse](#)

Clase [PutObjectRequest](#)

- [Espacio de nombres Amazon. KeyManagementService](#)

Clase [AmazonKeyManagementServiceClient](#)

- [Espacio de nombres Amazon. KeyManagementService](#).Modelo

Clase [CreateKeyRequest](#)

Clase [CreateKeyResponse](#)

El 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($" \n\nThe 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);
}
}
}
}

```

Consideraciones adicionales

- Puede comprobar los resultados de este ejemplo. Para ello, vaya a la [consola de Amazon S3](#) y abra el bucket que proporcionó a la aplicación. Luego, busque el nuevo objeto, descárguelo y ábralo en un editor de texto.
- La clase [AmazonS3 EncryptionClient V2](#) implementa la misma interfaz que la `AmazonS3Client` clase estándar. Esto facilita la migración del código a la clase `AmazonS3EncryptionClientV2` para que el cifrado y el descifrado se realicen de forma automática y transparente en el cliente.
- Una de las ventajas de usar una AWS KMS clave como clave maestra es que no necesita almacenar ni administrar sus propias claves maestras, ya que esto se consigue mediante AWS. Una segunda ventaja es que la `AmazonS3EncryptionClientV2` clase de AWS SDK para .NET es interoperable con la `AmazonS3EncryptionClientV2` clase de AWS SDK para Java. Esto significa que puede cifrar con el AWS SDK para Java y descifrar con el AWS SDK para .NET, y viceversa.

Note

La `AmazonS3EncryptionClientV2` clase de los AWS SDK para .NET admite claves maestras de KMS solo cuando se ejecuta en modo de metadatos. El modo de archivo de instrucciones de la `AmazonS3EncryptionClientV2` clase de AWS SDK para .NET es incompatible con la `AmazonS3EncryptionClientV2` clase de AWS SDK para Java.

- Para obtener más información sobre el cifrado del lado del cliente con la `AmazonS3EncryptionClientV2` clase y cómo funciona el cifrado de sobres, consulte [Cifrado de datos del lado del cliente con Amazon AWS SDK para .NET S3](#).

Envío de notificaciones desde la nube mediante Amazon Simple Notification Service

Note

La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK para .NET versión 3.3 y anteriores.

AWS SDK para .NET Es compatible con Amazon Simple Notification Service (Amazon SNS), que es un servicio web que permite a las aplicaciones, los usuarios finales y los dispositivos enviar notificaciones al instante desde la nube. Para obtener más información, consulte [Amazon SNS](#).

Enumeración de temas de Amazon SNS

En el siguiente ejemplo se muestra cómo enumerar sus temas de Amazon SNS, las suscripciones a cada tema y los atributos de cada tema. En este ejemplo se usa el valor predeterminado [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));
```

Envío de un mensaje a un tema de Amazon SNS

En el siguiente ejemplo se muestra cómo enviar un mensaje a un tema de Amazon SNS. El ejemplo toma un argumento, el ARN del tema de 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);
            }
        }
    }
}
```

```
}  
}
```

Consulte el [ejemplo completo](#), que incluye información sobre cómo compilar y ejecutar el ejemplo desde la línea de comandos, en adelante GitHub.

Envío de un mensaje SMS a un número de teléfono

En el siguiente ejemplo se muestra cómo enviar un mensaje SMS a un número de teléfono. El ejemplo acepta un argumento, el número de teléfono, que debe estar en cualquiera de los dos formatos descritos en los comentarios.

```
using System;  
using System.Linq;  
using System.Threading.Tasks;  
using Amazon;  
using Amazon.SimpleNotificationService;  
using Amazon.SimpleNotificationService.Model;  
  
namespace SnsPublish  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            // US phone numbers must be in the correct format:  
            // +1 (nnn) nnn-nnnn OR +1nnnnnnnnnn  
            string number = args[0];  
            string message = "Hello at " + DateTime.Now.ToShortTimeString();  
  
            var client = new AmazonSimpleNotificationServiceClient(region:  
Amazon.RegionEndpoint.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);
    }
}
}
```

Consulte el [ejemplo completo](#), que incluye información sobre cómo compilar y ejecutar el ejemplo desde la línea de comandos, en adelante GitHub.

Mensajería mediante Amazon SQS

AWS SDK para .NET Es compatible con [Amazon Simple Queue Service \(Amazon SQS\)](#), que [es un servicio](#) de cola de mensajes que gestiona los mensajes o los flujos de trabajo entre los componentes de un sistema.

Las colas de Amazon SQS proporcionan un mecanismo que permite enviar, almacenar y recibir mensajes entre componentes de software como, por ejemplo, microservicios, sistemas distribuidos y aplicaciones sin servidor. Esto permite desvincular esos componentes y acaba con la necesidad de tener que diseñar y operar un sistema de mensajería propio. Para obtener información sobre cómo funcionan las colas y los mensajes en Amazon SQS, consulte [Tutoriales de Amazon SQS](#) y [Funcionamiento de Amazon SQS](#) en la [Guía para desarrolladores de Amazon Simple Queue Service](#).

Important

Dada la naturaleza distribuida de las colas, Amazon SQS no puede garantizar que vaya a recibir los mensajes en el mismo orden en que se envían. Si necesita conservar el orden de los mensajes, utilice una [cola FIFO de Amazon SQS](#).

APIs

Se AWS SDK para .NET proporciona APIs para los clientes de Amazon SQS. Le APIs permiten trabajar con funciones de Amazon SQS, como colas y mensajes. Esta sección contiene una pequeña cantidad de ejemplos que le muestran los patrones que puede seguir al trabajar con ellos. APIs

Para ver el conjunto completo de ellos APIs, consulta la [referencia de la AWS SDK para .NET API](#) (y desplázate hasta «Amazon.sqs»).

Los Amazon SQS APIs se proporcionan en el paquete [AWSSDK NuGet .SQS](#).

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el](#) proyecto. Revise también la información en [Uso del SDK](#).

Temas

Temas

- [Creación de colas de Amazon SQS](#)
- [Actualización de colas de Amazon SQS](#)
- [Eliminación de colas de Amazon SQS](#)
- [Envío de mensajes de Amazon SQS](#)
- [Recepción de mensajes de Amazon SQS](#)

Creación de colas de Amazon SQS

En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para crear una cola de Amazon SQS. La aplicación crea una [cola de mensajes fallidos](#) si no se proporciona el ARN de una. A continuación, crea una cola de mensajes estándar, que incluye una cola de mensajes fallidos (una que haya proporcionado o que se haya creado).

Si no proporciona ningún argumento de línea de comandos, la aplicación simplemente muestra información de todas las colas existentes.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Visualización de las colas existentes](#)
- [Creación de la cola](#)
- [Obtención del ARN de una cola](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Visualización de las colas existentes

El siguiente fragmento de código muestra una lista de las colas existentes en la región del cliente de SQS, así como los atributos de cada cola.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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}");
}
```

Creación de la cola

El siguiente fragmento de código crea una cola. El fragmento de código incluye el uso de una cola de mensajes fallidos, pero esta no tiene por qué ser necesaria para sus colas.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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;
}

```

Obtención del ARN de una cola

El siguiente fragmento de código obtiene el ARN de la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.SQS](#)

Elementos de programación:

- Espacio de nombres [Amazon.SQS](#)

Clase [Amazon SQSClient](#)

Clase [QueueAttributeName](#)

- Espacio de nombres [Amazon.SQS.Model](#)

Clase [CreateQueueRequest](#)

Clase [CreateQueueResponse](#)

Clase [GetQueueAttributesResponse](#)

Clase [ListQueuesResponse](#)

El 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($"{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);
    }
}
```

Consideraciones adicionales

- El nombre de la cola debe estar formado solo por caracteres alfanuméricos, guiones y guiones bajos.
- Los nombres de las colas y las colas distinguen mayúsculas URLs de minúsculas
- Si necesita la URL de cola, pero solo tiene el nombre de la cola, utilice uno de los métodos `AmazonSQSClient.GetQueueUrlAsync`.
- Para obtener información sobre los distintos atributos de cola que puede configurar, consulte la referencia de [CreateQueueRequest](#) la [AWS SDK para .NET API](#) o la referencia de [SetQueueAttributes](#) la [API de Amazon Simple Queue Service](#).
- En este ejemplo se especifica un sondeo prolongado de todos los mensajes de la cola creada. Esto se realiza mediante el atributo `ReceiveMessageWaitTimeSeconds`.

También puedes especificar un sondeo prolongado durante una llamada a los `ReceiveMessageAsync` métodos de la `SQSClient` clase [Amazon](#). Para obtener más información, consulte [Recepción de mensajes de Amazon SQS](#).

Para obtener información sobre los sondeos cortos frente a los sondeos largos, consulte [Sondeos cortos y largos](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

- Una cola de mensajes fallidos es una cola a la que otras colas (de origen) pueden enviar mensajes que no se han procesado correctamente. Para obtener más información, consulte [Colas de mensajes fallidos de Amazon SQS](#) en la Guía para desarrolladores de Amazon Simple Queue Service.
- La lista de colas y los resultados de este ejemplo también se pueden ver en la [consola de Amazon SQS](#).

Actualización de colas de Amazon SQS

En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para actualizar una cola de Amazon SQS. Tras realizar algunas comprobaciones, la aplicación actualiza el atributo especificado con el valor indicado y, a continuación, muestra todos los atributos de la cola.

Si solo se incluye la URL de cola en los argumentos de la línea de comandos, la aplicación simplemente muestra todos los atributos de la cola.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Visualización de atributos de cola](#)
- [Validación del nombre de atributo](#)
- [Actualización de un atributo de cola](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Visualización de atributos de cola

El siguiente fragmento de código muestra los atributos de la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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}");  
}
```

Validación del nombre de atributo

El siguiente fragmento de código valida el nombre del atributo que se está actualizando.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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;  
}
```

Actualización de un atributo de cola

El siguiente fragmento de código actualiza un atributo de la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.SQS](#)

Elementos de programación:

- Espacio de nombres [Amazon.SQS](#)

Clase [Amazon SQSClient](#)

Clase [QueueAttributeName](#)

- Espacio de nombres [Amazon.SQS.Model](#)

Clase [GetQueueAttributesResponse](#)

El 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($"\\nThe given attribute name, {attribute}, isn't valid.");
    }
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
            new List<string>{ QueueAttributeName.All });
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\\t{att.Key}: {att.Value}");
}

//
// Method to check the name of the attribute
private static bool ValidAttribute(string attribute)
{
    var attOk = false;
    var qAttNameType = typeof(QueueAttributeName);
    List<string> qAttNamefields = new List<string>();
    foreach(var field in qAttNameType.GetFields())
        qAttNamefields.Add(field.Name);
    foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
    return attOk;
}

//
// Method to update a queue attribute
private static async Task UpdateAttribute(
    IAmazonSQS sqsClient, string qUrl, string attribute, string value)
{
    await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
}

```

```

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
    Console.WriteLine("  -q: The URL of the queue you want to update.");
    Console.WriteLine("  -a: The name of the attribute to update.");
    Console.WriteLine("  -v, --value: The value to assign to the attribute.");
}
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    // (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    // where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
        var parsedArgs = new Dictionary<string,string>();
        int i = 0, n = 0;
        while(i < args.Length)
        {
            // If the first argument in this iteration starts with a dash it's an option.
            if(args[i].StartsWith("-"))
            {
                var key = args[i++];
                var value = key;

```



```

        // Check to see if there's a value that goes with this option?
        if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
        parsedArgs.Add(key, value);
    }

    // If the first argument in this iteration doesn't start with a dash, it's a
value
    else
    {
        parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
        n++;
    }
}

return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

//
// Method to exit the application with an error.
public static void ErrorExit(string msg, int code=1)
{
    Console.WriteLine("\nError");
    Console.WriteLine(msg);
    Environment.Exit(code);
}

```

```
}  
  
}
```

Consideraciones adicionales

- Para actualizar el `RedrivePolicy` atributo, debe citar el valor completo y evitar las comillas de los key/value pares, según corresponda a su sistema operativo.

Por ejemplo, en Windows el valor se crea de forma similar a la siguiente:

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\", \"maxReceiveCount\":\"10\"}"
```

Eliminación de colas de Amazon SQS

En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para eliminar una cola de Amazon SQS. La aplicación elimina la cola, espera un periodo de tiempo determinado a que desaparezca y, a continuación, muestra una lista de las colas restantes.

Si no proporciona ningún argumento de línea de comandos, la aplicación simplemente muestra una lista de las colas existentes.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Eliminación de la cola](#)
- [Espera para que la cola desaparezca](#)
- [Visualización de una lista de colas existentes](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Eliminación de la cola

El siguiente fragmento de código elimina la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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.");  
}
```

Espera para que la cola desaparezca

El siguiente fragmento de código espera a que finalice el proceso de eliminación, que puede tardar 60 segundos.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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;  
    }  
}
```

```
}
```

Visualización de una lista de colas existentes

El siguiente fragmento de código muestra una lista de las colas existentes en la región del cliente de SQS.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.SQS](#)

Elementos de programación:

- Espacio de nombres [Amazon.SQS](#)
 - Clase [Amazon SQSClient](#)
- Espacio de nombres [Amazon.SQS.Model](#)
 - Clase [ListQueuesResponse](#)

El 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}");
}
}
```

Consideraciones adicionales

- La llamada a la API `DeleteQueueAsync` no comprueba si la cola que se va a eliminar se está usando como cola de mensajes fallidos. Esto podría comprobarse con un procedimiento más elaborado.
- La lista de colas y los resultados de este ejemplo también se pueden ver en la [consola de Amazon SQS](#).

Envío de mensajes de Amazon SQS

[En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para enviar mensajes a una cola de Amazon SQS, que puede crear mediante programación o mediante la consola de Amazon SQS.](#)

La aplicación envía un único mensaje a la cola y, a continuación, un lote de mensajes. Después, la aplicación espera a que el usuario especifique información, que pueden ser más mensajes para enviarlos a la cola o una solicitud para salir de la aplicación.

Este ejemplo y el [siguiente, que ilustra cómo recibir mensajes](#), se pueden usar juntos para ver el flujo de mensajes en Amazon SQS.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Enviar un mensaje](#)

- [Envío de un lote de mensajes](#)
- [Eliminación de todos los mensajes de la cola](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Enviar un mensaje

El siguiente fragmento de código envía un mensaje a la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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}");  
}
```

Envío de un lote de mensajes

El siguiente fragmento de código envía un lote de mensajes a la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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.");  
    }
```

Eliminación de todos los mensajes de la cola

El siguiente fragmento de código elimina todos los mensajes de la cola identificada por la URL de cola especificada. Esto se conoce también como purgar la cola.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.SQS](#)

Elementos de programación:

- Espacio de nombres [Amazon.SQS](#)

Clase [Amazon SQSClient](#)

- Espacio de nombres [Amazon.SQS.Model](#)

Clase [PurgeQueueResponse](#)

Clase [SendMessageBatchResponse](#)

Clase [SendMessageResponse](#)

Clase [SendMessageBatchRequestEntry](#)

Clase [SendMessageBatchResultEntry](#)

El 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 = "{\"product\":{\"name\":\"Product A\",\"price\": \"32\"},{\"name\": \"Product B\",\"price\": \"27\"}}";
        private const string XmlMessage = "<products><product name=\"Product A\" price= \"32\" /><product name=\"Product B\" price=\"27\" /></products>";
        private const string CustomMessage = "||product|Product A|32||product|Product B|27||";
        private const string TextMessage = "Just a plain text message.";

        static async Task Main(string[] args)
        {
            // Do some checks on the command-line
            if(args.Length == 0)
            {
                Console.WriteLine("\nUsage: SQSSendMessages queue_url");
                Console.WriteLine("    queue_url - The URL of an existing SQS queue.");
                return;
            }
            if(!args[0].StartsWith("https://sqs."))
            {
                Console.WriteLine("\nThe command-line argument isn't a queue URL:");
            }
        }
    }
}
```

```

        Console.WriteLine($"{args[0]}");
        return;
    }

    // Create the Amazon SQS client
    var sqsClient = new AmazonSQSClient();

    // (could verify that the queue exists)
    // Send some example messages to the given queue
    // A single message
    await SendMessage(sqsClient, args[0], JsonMessage);

    // A batch of messages
    var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
    await SendMessageBatch(sqsClient, args[0], batchMessages);

    // Let the user send their own messages or quit
    await InteractWithUser(sqsClient, args[0]);

    // Delete all messages that are still in the queue
    await DeleteAllMessages(sqsClient, args[0]);
}

//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
{
    SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}

//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects

```

```

private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
    Console.WriteLine($"\\nSending a batch of messages to queue\\n {qUrl}");
    SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
}

//
// Method to get input from the user
// They can provide messages to put in the queue or exit the application
private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
{
    string response;
    while (true)
    {
        // Get the user's input
        Console.WriteLine("\\nType a message for the queue or \\\"exit\\\" to quit:");
        response = Console.ReadLine();
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
    }
}

//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($"\\nPurging messages from queue\\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
}
}

```

Consideraciones adicionales

- Para obtener información sobre las diferentes limitaciones de los mensajes, incluidos los caracteres permitidos, consulte [Cuotas de los mensajes](#) en la [Guía para desarrolladores de Amazon Simple Queue Service](#).
- Los mensajes permanecen en las colas hasta que se eliminan o hasta que la cola se purga. Cuando una aplicación ha recibido un mensaje, este no estará visible en la cola aunque siga existiendo en ella. Para obtener más información sobre los tiempos de espera de visibilidad, consulte [Tiempo de espera de visibilidad de Amazon SQS](#).
- Además del cuerpo del mensaje, también se pueden agregar atributos a los mensajes. Para obtener más información, consulte [Metadatos de mensajes](#).

Recepción de mensajes de Amazon SQS

[En este ejemplo, se muestra cómo utilizarla AWS SDK para .NET para recibir mensajes de una cola de Amazon SQS, que puede crear mediante programación o mediante la consola de Amazon SQS](#). La aplicación lee un solo mensaje de la cola, lo procesa (en este caso, muestra el cuerpo del mensaje en la consola) y, a continuación, lo elimina de la cola. La aplicación repite estos pasos hasta que el usuario pulse una tecla del teclado.

Este ejemplo y el [anterior, que ilustra cómo enviar mensajes](#), se pueden usar juntos para ver el flujo de mensajes en Amazon SQS.

En las siguientes secciones se proporcionan fragmentos de código de este ejemplo. Tras ello, se muestra el [código completo del ejemplo](#), que se puede compilar y ejecutar tal cual.

Temas

- [Recepción de un mensaje](#)
- [Eliminación de un mensaje](#)
- [Código completo](#)
- [Consideraciones adicionales](#)

Recepción de un mensaje

El siguiente fragmento de código recibe un mensaje de la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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.)  
    });  
}
```

Eliminación de un mensaje

El siguiente fragmento de código elimina un mensaje de la cola identificada por la URL de cola especificada.

El ejemplo que aparece [al final de este tema](#) muestra este fragmento de código en 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

En esta sección se muestran las referencias relevantes y el código completo de este ejemplo.

Referencias de SDK

NuGet paquetes:

- [AWSSDK.SQS](#)

Elementos de programación:

- Espacio de nombres [Amazon.SQS](#)

Clase [Amazon SQSClient](#)

- Espacio de nombres [Amazon.SQS.Model](#)

Clase [ReceiveMessageRequest](#)

Clase [ReceiveMessageResponse](#)

El 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);
    }
}
```

Consideraciones adicionales

- Para especificar un sondeo prolongado, en este ejemplo se usa la propiedad `WaitTimeSeconds` en cada llamada al método `ReceiveMessageAsync`.

También se puede especificar un sondeo prolongado para todos los mensajes de una cola usando el atributo `ReceiveMessageWaitTimeSeconds` al [crear](#) o [actualizar](#) la cola.

Para obtener información sobre los sondeos cortos frente a los sondeos largos, consulte [Sondeos cortos y largos](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

- Durante el procesamiento de los mensajes, puede utilizar el controlador de recepción para cambiar el tiempo de espera de visibilidad de los mensajes. Para obtener información sobre cómo hacerlo, consulta los `ChangeMessageVisibilityAsync` métodos de la `SQSClient` clase [Amazon](#).
- Cuando se llama al método `DeleteMessageAsync` sin condiciones, el mensaje se elimina de la cola, sin importar la configuración del tiempo de espera de visibilidad.

Uso AWS Lambda para servicio de cómputo

El AWS SDK para .NET soporta AWS Lambda, que le permite ejecutar código sin aprovisionar ni administrar servidores. Para obtener más información, consulte la [página del producto AWS Lambda](#) y la [Guía para desarrolladores de AWS Lambda](#), especialmente la sección [Trabajar con C#](#).

APIs

El AWS SDK para .NET APIs prevé. AWS Lambda [Le APIs permiten trabajar con funciones de Lambda, como funciones, activadores y eventos](#). Para ver el conjunto completo de APIs, consulte [Lambda](#) en la referencia de la [AWS SDK para .NET API](#).

[Las Lambda APIs se proporcionan en paquetes. NuGet](#)

Requisitos previos

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información en [Uso del SDK](#).

Información adicional

Consulte [Integración AWS con .NET Aspire](#) para obtener información sobre cómo desarrollar AWS Lambda con .NET Aspire.

Temas

Temas

- [Uso de anotaciones para escribir funciones AWS Lambda](#)

Uso de anotaciones para escribir funciones AWS Lambda

Al escribir funciones de Lambda, a veces es necesario escribir una gran cantidad de código de controlador y actualizar plantillas de AWS CloudFormation, entre otras tareas. Lambda Annotations es un marco que ayuda a aliviar estas cargas con las funciones de Lambda de .NET 6, lo que hace que la experiencia de escribir Lambda resulte más natural en C#.

Considere los siguientes fragmentos de código, en los que se suman dos números, como ejemplo de las ventajas de utilizar el marco Lambda Annotations.

Sin 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" } }
        };
    }
}
```

Con Lambda Annotations

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
```

```
        return x + y;  
    }  
}
```

Como se muestra en el ejemplo, Lambda Annotations permite prescindir de tener que usar código repetitivo en cierta medida.

Para saber cómo utilizar el marco, así como obtener información adicional, consulte los siguientes recursos:

- El [GitHub archivo README](#) para obtener documentación sobre las APIs anotaciones Lambda y sus atributos.
- La [entrada de blog](#) de Lambda Annotations.
- El paquete [Amazon.Lambda.Annotations](#) NuGet .
- El proyecto [Photo Asset Management está](#) en marcha. GitHub [En concreto, consulte la PamApiAnnotationscarpeta y las referencias a las anotaciones Lambda en el archivo README del proyecto.](#)

Note

El ejemplo anterior es específico de la versión 3 del. AWS SDK para .NET Si usa el ejemplo con la versión 4 del SDK (la versión más reciente), es posible que deba realizar ajustes de acuerdo con la información que contiene. [Migración a la versión 4](#)

Bibliotecas y marcos de alto nivel para AWS SDK para .NET

Las siguientes secciones contienen información sobre bibliotecas y marcos de alto nivel que no forman parte de la funcionalidad principal del SDK. Estas bibliotecas y marcos utilizan la funcionalidad principal del SDK para crear funciones que faciliten determinadas tareas.

Si eres nuevo en el tema AWS SDK para .NET, quizás quieras consultar primero el [Crear una aplicación sencilla](#) tema. que sirve de introducción a SDK.

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información en [Uso del SDK](#).

Temas

- [AWS Marco de procesamiento de mensajes para.NET](#)
- [Integración AWS con .NET Aspire en AWS SDK para .NET](#)

AWS Marco de procesamiento de mensajes para.NET

El marco de procesamiento de AWS mensajes para.NET es un marco AWS nativo que simplifica el desarrollo de aplicaciones de procesamiento de mensajes de.NET que utilizan AWS servicios como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) y Amazon EventBridge. El marco reduce la cantidad de código repetitivo que deben escribir los desarrolladores, lo que le permite centrarse en la lógica empresarial a la hora de publicar y consumir mensajes. Para obtener más información sobre cómo el marco puede simplificar su desarrollo, consulte la entrada del blog [Introducción al marco de procesamiento de AWS mensajes para .NET](#) (versión preliminar). La primera parte, en particular, ofrece una demostración que muestra la diferencia entre usar llamadas a la API de bajo nivel y usar el marco.

El marco de procesamiento de mensajes admite las siguientes actividades y funciones:

- Enviar mensajes a SQS y publicar eventos en SNS y EventBridge
- Recibir y gestionar mensajes de SQS mediante un sondeador de larga duración, que normalmente se utiliza en los servicios en segundo plano. Esto incluye administrar el tiempo de espera de visibilidad mientras se está gestionando un mensaje para evitar que otros clientes lo procesen.
- Gestión de los mensajes en AWS Lambda las funciones.
- Colas de SQS de FIFO (first-in-first-out) y temas de SNS.
- OpenTelemetry para el registro.

Para obtener más información sobre estas actividades y funciones, consulte la sección Características de la entrada del [blog](#) y los temas que se enumeran a continuación.

Antes de empezar, asegúrese de haber [configurado el entorno y el proyecto](#). Revise también la información en [Uso del SDK](#).

Recursos adicionales

- El [AWS.Messaging](#) paquete está en [NuGet.org](#).
- La [referencia de la API](#).
- El README archivo del GitHub repositorio en <https://github.com/aws/aws-dotnet-messaging/>

- [Inyección de dependencias de.NET](#) de Microsoft.
- [.NET Generic Host](#) de Microsoft.

Temas

- [Comience con el marco de procesamiento de AWS mensajes para.NET](#)
- [Publica mensajes con el marco de procesamiento de AWS mensajes para.NET](#)
- [Consume mensajes con el marco de procesamiento de AWS mensajes para.NET](#)
- [Uso de FIFO con el marco de procesamiento de AWS mensajes para.NET](#)
- [Registro y telemetría abierta para el marco de procesamiento de AWS mensajes para.NET](#)
- [Personalice el marco de procesamiento de AWS mensajes para.NET](#)
- [Seguridad del marco de procesamiento de AWS mensajes para.NET](#)

Comience con el marco de procesamiento de AWS mensajes para.NET

Antes de empezar, asegúrese de haber [configurado su entorno y su proyecto](#). Revise también la información en [Uso del SDK](#).

En este tema se proporciona información que le ayudará a empezar a utilizar el marco de procesamiento de mensajes. Además de la información sobre los requisitos previos y la configuración, se proporciona un tutorial que muestra cómo implementar un escenario común.

Requisitos previos y configuración

- Las credenciales que proporcione para la aplicación deben tener los permisos adecuados para el servicio de mensajería y las operaciones que utilice. Para obtener más información, consulte los temas de seguridad de [SQS](#) y [SNS](#) y sus respectivas [EventBridge](#) guías para desarrolladores. [Consulte también la parte del archivo README GitHub que trata sobre los permisos específicos.](#)
- Para usar el marco de procesamiento de AWS mensajes para .NET, debe agregar el [AWS.Messaging](#) NuGetpaquete a su proyecto. Por ejemplo:

```
dotnet add package AWS.Messaging
```

- El marco se integra con el [contenedor de servicios de inyección de dependencias \(DI\) de .NET](#). Puede configurar el marco durante el inicio de la aplicación llamándolo `AddAWSMessageBus` para agregarlo al contenedor 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

En este tutorial se muestra cómo utilizar el marco de procesamiento de AWS mensajes para.NET. Crea dos aplicaciones: una API mínima de ASP.NET Core que envía mensajes a una cola de Amazon SQS cuando recibe una solicitud en un punto final de la API, y una aplicación de consola de larga ejecución que sondea estos mensajes y los gestiona.

- Las instrucciones de este tutorial favorecen la CLI de.NET, pero puede realizar este tutorial mediante herramientas multiplataforma, como la CLI de.NET o Microsoft Visual Studio. Para obtener información acerca de las herramientas, consulte [Instalación y configuración de la cadena de herramientas para AWS SDK para .NET](#).
- En este tutorial se da por sentado que está utilizando su [default] perfil como credenciales. También supone que las credenciales de corta duración están disponibles con los permisos adecuados para enviar y recibir mensajes de Amazon SQS. Para obtener más información, consulte [Autenticar el con AWS SDK para .NET AWS](#) los temas de seguridad de [SQS](#).

Note

Si ejecuta este tutorial, podría incurrir en costes por la mensajería de SQS.

Pasos

- [Cree una cola de SQS](#)
- [Cree y ejecute la aplicación de publicación](#)
- [Cree y ejecute la aplicación de manejo](#)

- [Limpieza](#)

Cree una cola de SQS

Este tutorial requiere una cola SQS para enviar y recibir mensajes. Se puede crear una cola mediante uno de los siguientes comandos para el o el AWS CLI . Herramientas de AWS para PowerShell Tome nota de la URL de la cola que se devuelve para poder especificarla en la siguiente configuración estructural.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Herramientas de AWS para PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Cree y ejecute la aplicación de publicación

Utilice el siguiente procedimiento para crear y ejecutar la aplicación de publicación.

1. Abra una línea de comandos o un terminal. Busque o cree una carpeta del sistema operativo en la que pueda crear un proyecto .NET.
2. En esa carpeta, ejecute el siguiente comando para crear el proyecto .NET.

```
dotnet new webapi --name Publisher
```

3. Navega hasta la carpeta del nuevo proyecto. Agregue una dependencia al marco de procesamiento de AWS mensajes para .NET.

```
cd Publisher  
dotnet add package AWS.Messaging
```

Note

Si lo utiliza AWS IAM Identity Center para la autenticación, asegúrese de añadir también `AWSSDK.SSO` y `AWSSDK.SSO0IDC`.

4. Sustituya el código Program.cs por el siguiente código.

```
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. Ejecute el siguiente comando. Esto debería abrir una ventana del navegador con la interfaz de usuario de Swagger, que le permitirá explorar y probar su API.

```
dotnet watch run <queue URL created earlier>
```

6. Abra el `/greeting` punto final y elige Pruébalo.
7. Especifique `senderName` los `greeting` valores del mensaje y elija Ejecutar. Esto invoca su API, que envía el mensaje SQS.

Cree y ejecute la aplicación de manejo

Utilice el siguiente procedimiento para crear y ejecutar la aplicación de manipulación.

1. Abra una línea de comandos o un terminal. Busque o cree una carpeta del sistema operativo en la que pueda crear un proyecto .NET.
2. En esa carpeta, ejecute el siguiente comando para crear el proyecto .NET.

```
dotnet new console --name Handler
```

3. Navega hasta la carpeta del nuevo proyecto. Agregue una dependencia al marco de procesamiento de AWS mensajes para .NET. Agregue también el `Microsoft.Extensions.Hosting` paquete, que le permite configurar el marco a través del [host genérico.NET](#).

```
cd Handler  
dotnet add package AWS.Messaging  
dotnet add package Microsoft.Extensions.Hosting
```

Note

Si lo utiliza AWS IAM Identity Center para la autenticación, asegúrese de añadir también `AWSSDK.SSO` y `AWSSDK.SSO00IDC`.

4. Sustituya el código `Program.cs` por el siguiente código.

```

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. Ejecute el siguiente comando. Esto inicia un sondeo de larga duración.

```
dotnet run <queue URL created earlier>
```

Poco después del inicio, la aplicación recibirá el mensaje que se envió en la primera parte de este tutorial y registrará el siguiente mensaje:

```
Received message {greeting} from {senderName}
```

6. Pulse Ctrl+C para detener el sondeo.

Limpieza

Utilice uno de los siguientes comandos para eliminar la cola AWS CLI o Herramientas de AWS para PowerShell para eliminarla.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Herramientas de AWS para PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publica mensajes con el marco de procesamiento de AWS mensajes para .NET

El marco de procesamiento de AWS mensajes para .NET permite publicar uno o más tipos de mensajes, procesar uno o más tipos de mensajes o hacer ambas cosas en la misma aplicación.

El código siguiente muestra la configuración de una aplicación que publica distintos tipos de mensajes en distintos AWS servicios.

```
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");
});
```

Una vez que haya registrado el marco durante el inicio, introduzca el genérico `IMessagePublisher` en el código. Llame a su `PublishAsync` método para publicar cualquiera de los tipos de mensajes que se configuraron anteriormente. El publicador genérico determinará el destino al que se debe enrutar el mensaje en función de su tipo.

En el siguiente ejemplo, un controlador MVC de ASP.NET recibe `ChatMessage` mensajes y `OrderInfo` eventos de los usuarios y, a continuación, los publica en Amazon SQS y Amazon SNS, respectivamente. Ambos tipos de mensajes se pueden publicar con el editor genérico que se configuró anteriormente.

```
[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 dirigir un mensaje a la lógica de gestión adecuada, el marco utiliza metadatos denominados identificador del tipo de mensaje. De forma predeterminada, es el nombre completo del tipo.NET del mensaje, incluido su nombre de ensamblado. Si envía y gestiona mensajes a la vez, este mecanismo funciona bien si comparte la definición de los objetos de mensaje en todos los proyectos. Sin embargo, si los mensajes se redefinen en espacios de nombres diferentes o si intercambia mensajes con otros marcos o lenguajes de programación, es posible que tengas que anular el identificador del tipo de mensaje.

```
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");
    });
});
```

Editores de servicios específicos

El ejemplo que se muestra arriba usa el genérico `IMessagePublisher`, que puede publicarse en cualquier AWS servicio compatible en función del tipo de mensaje configurado. El marco también proporciona editores de servicios específicos para Amazon SQS, Amazon SNS y Amazon EventBridge. Estos editores específicos exponen opciones que solo se aplican a ese servicio y se pueden insertar mediante los tipos `ISQSPublisher`, `ISNSPublisher` y `IEventBridgePublisher`.

Por ejemplo, al enviar mensajes a una cola FIFO de SQS, debe establecer el ID de grupo de [mensajes](#) correspondiente. El código siguiente vuelve a mostrar el `ChatMessage` ejemplo, pero ahora se utiliza una `ISQSPublisher` para configurar las opciones específicas de 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();
}
}

```

Se puede hacer lo mismo con SNS y EventBridge, utilizando `ISNSPublisher` y respectivamente. `IEventBridgePublisher`

```

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>
});
```

De forma predeterminada, los mensajes de un tipo determinado se envían al destino que se haya configurado de antemano. Sin embargo, puede anular el destino de un solo mensaje utilizando los editores específicos del mensaje. También puede anular el AWS SDK para .NET cliente subyacente que se utiliza para publicar el mensaje, lo que puede resultar útil en aplicaciones con varios usuarios en las que es necesario cambiar las funciones o las credenciales, según el destino.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consume mensajes con el marco de procesamiento de AWS mensajes para.NET

El marco de procesamiento de AWS mensajes para .NET permite consumir los mensajes que se han [publicado](#) mediante el marco o uno de los servicios de mensajería. Los mensajes se pueden consumir de diversas formas, algunas de las cuales se describen a continuación.

Controladores de mensajes

Para consumir mensajes, implemente un controlador de mensajes mediante la `IMessageHandler` interfaz de cada tipo de mensaje que desee procesar. El mapeo entre los tipos de mensajes y los controladores de mensajes se configura al inicio del proyecto.

```
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();

```

El siguiente código muestra un ejemplo de controlador de mensajes para un ChatMessage mensaje.

```

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());
    }
}

```

El exterior `MessageEnvelope` contiene los metadatos utilizados por el marco. Su `message` propiedad es el tipo de mensaje (en este caso `ChatMessage`).

Puede volver `MessageProcessStatus.Success()` a indicar que el mensaje se ha procesado correctamente y el marco eliminará el mensaje de la cola de Amazon SQS. Cuando regrese `MessageProcessStatus.Failed()`, el mensaje permanecerá en la cola, donde podrá volver a procesarse o pasarse a una cola con [letra muerta, si está configurada](#).

Manejo de mensajes en un proceso prolongado

Puede llamar `AddSQSPoller` con una URL de cola de SQS para iniciar una sesión prolongada [BackgroundService](#) que sondee continuamente la cola y procese los mensajes.

```
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();
```

Configuración del sondeador de mensajes de SQS

El sondeador de mensajes SQS se puede configurar al llamar `SQSMessagePollerOptions.AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- El número máximo de mensajes de la cola que se pueden procesar simultáneamente. El valor predeterminado es 10.
- `WaitTimeSeconds`- El tiempo (en segundos) durante el que la llamada `ReceiveMessage` SQS espera a que un mensaje llegue a la cola antes de volver a recibirlo. Si hay un mensaje disponible, la llamada se devuelve antes de eso. `WaitTimeSeconds` El valor predeterminado es 20.

Control del tiempo de espera de visibilidad de los mensajes

Los mensajes de SQS tienen un período de tiempo de [espera de visibilidad](#). Cuando un consumidor comienza a gestionar un mensaje determinado, permanece en la cola pero se oculta al resto de los consumidores para evitar que lo procesen más de una vez. Si el mensaje no se gestiona y se elimina antes de volver a ser visible, es posible que otro consumidor intente gestionar el mismo mensaje.

El marco realizará un seguimiento del tiempo de espera de visibilidad de los mensajes que gestiona actualmente e intentará ampliarlo. Puede configurar este comportamiento `SQSMessagePollerOptions` al llamar `AddSQSPoller`.

- `VisibilityTimeout`- El tiempo en segundos que los mensajes recibidos permanecen ocultos para las solicitudes de recuperación posteriores. El valor predeterminado es 30.
- `VisibilityTimeoutExtensionThreshold`- Cuando el tiempo de espera de visibilidad de un mensaje esté dentro de estos segundos desde su fecha de caducidad, el marco ampliará el tiempo de espera de visibilidad (unos `VisibilityTimeout` segundos más). El valor predeterminado es 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Con qué frecuencia, en cuestión de segundos, el marco comprobará si hay mensajes que estén a `VisibilityTimeoutExtensionThreshold` unos segundos de caducar y, a continuación, ampliará su tiempo de espera de visibilidad. El valor predeterminado es 1.

En el siguiente ejemplo, el marco comprobará cada segundo si hay mensajes que aún se estén gestionando. En el caso de los mensajes transcurridos 5 segundos desde que vuelvan a ser visibles, el marco ampliará automáticamente el tiempo de espera de visibilidad de cada mensaje otros 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;
    });
```

Gestión de los mensajes en las funciones AWS Lambda

Puede usar el marco de procesamiento de AWS mensajes para.NET con la [integración de SQS con Lambda](#). Esto lo proporciona el `AWS.Messaging.Lambda` paquete. Consulte su [archivo README](#) para empezar.

Uso de FIFO con el marco de procesamiento de AWS mensajes para.NET

Para los casos de uso en los que el orden y la deduplicación de mensajes son fundamentales, el marco de procesamiento de AWS mensajes para .NET admite las colas de [Amazon SQS first-in-first-out](#) (FIFO) y los temas de [Amazon](#) SNS.

Publicación

Al publicar mensajes en una cola o tema de la FIFO, debe establecer el ID del grupo de mensajes, que especifica el grupo al que pertenece el mensaje. Los mensajes de un grupo se procesan en orden. Puede configurarlo en los editores de mensajes específicos de SQL y de SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Suscripción

Al gestionar los mensajes de una cola FIFO, el marco gestiona los mensajes de un grupo de mensajes determinado en el orden en que se recibieron para cada llamada. `ReceiveMessages` El marco entra en este modo de operación automáticamente cuando se configura con una cola que termina en `.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 y telemetría abierta para el marco de procesamiento de AWS mensajes para.NET

El marco de procesamiento de AWS mensajes para .NET está diseñado OpenTelemetry para registrar los [rastros](#) de cada mensaje publicado o gestionado por el marco. Esto lo proporciona el [AWS.Messaging.Telemetry.OpenTelemetry](#) paquete. Consulte su [archivo README](#) para empezar.

Note

Para obtener información de seguridad relacionada con el registro, consulte [Seguridad del marco de procesamiento de AWS mensajes para.NET](#).

Personalice el marco de procesamiento de AWS mensajes para.NET

El marco de procesamiento de AWS mensajes para .NET crea, envía y gestiona los mensajes en tres «capas» diferentes:

1. En la capa más externa, el marco crea la solicitud o respuesta AWS nativa específica de un servicio. Con Amazon SQS, por ejemplo, crea [SendMessage](#) solicitudes y trabaja con los [Message](#) objetos definidos por el servicio.
2. [Dentro de la solicitud y respuesta de SQS, el marco establece el MessageBody elemento \(o Message para Amazon SNS Detail o EventBridge Amazon\) en formato JSON. CloudEvent](#)

Contiene los metadatos establecidos por el marco a los que se puede acceder en el `MessageEnvelope` objeto cuando se gestiona un mensaje.

3. En la capa más interna, el `data` atributo del objeto `CloudEvent` JSON contiene una serialización en JSON del objeto .NET que se envió o recibió como mensaje.

```
{
  "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>"
}
```

Puede personalizar la forma en que se configura y lee el sobre del mensaje:

- `"id"` identifica el mensaje de forma exclusiva. De forma predeterminada, se establece en un nuevo GUID, pero esto se puede anular implementando el suyo propio `IMessageIdGenerator` e inyectándolo en el contenedor del DI.
- `"type"` controla cómo se enruta el mensaje a los controladores. De forma predeterminada, utiliza el nombre completo del tipo .NET que corresponde al mensaje. Puede anularlo mediante el `messageTypeIdentifier` parámetro al asignar el tipo de mensaje al destino mediante `AddSQSPublisher`, `AddSNSPublisher`, o `AddEventBridgePublisher`.
- `"source"` indica qué sistema o servidor envió el mensaje.
 - Este será el nombre de la función si se publica desde AWS Lambda, el nombre del clúster y el ARN de la tarea si está en Amazon ECS, el ID de la instancia si está en Amazon; de lo contrario EC2, será un valor alternativo de `/aws/messaging`
 - Puede anular esto a través de `AddMessageSource` o `AddMessageSourceSuffix` en `MessageBusBuilder`
- `"time"` establecido en el valor actual `DateTime` en UTC. Esto se puede anular implementando el suyo `IDateTimeHandler` e inyectándolo en el contenedor DI.
- `"data"` contiene una representación en JSON del objeto .NET que se envió o recibió como mensaje:
 - `ConfigureSerializationOptions` on `MessageBusBuilder` le permite configurar el [System.Text.Json.JsonSerializerOptions](#) que se utilizará al serializar y deserializar el mensaje.

- Para añadir atributos adicionales o transformar el sobre del mensaje una vez que el framework lo haya creado, puede implementarlos `ISerializationCallback` y registrarlos mediante `on.AddSerializationCallback MessageBusBuilder`

Seguridad del marco de procesamiento de AWS mensajes para .NET

El marco de procesamiento de AWS mensajes de .NET se basa en el AWS SDK para .NET para comunicarse con AWS. Para obtener más información sobre la seguridad en AWS SDK para .NET, consulte [Seguridad para este AWS producto o servicio](#).

Por motivos de seguridad, el marco no registra los mensajes de datos enviados por el usuario. Si desea habilitar esta funcionalidad con fines de depuración, debe llamar `EnableDataMessageLogging()` al bus de mensajes de la siguiente manera:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Si descubre un posible problema de seguridad, consulte la [política de seguridad](#) para obtener más información.

Integración AWS con .NET Aspire en AWS SDK para .NET

.NET Aspire es una nueva forma de crear aplicaciones listas para la nube. En particular, proporciona una organización para los entornos locales en la que ejecutar, conectar y depurar los componentes de las aplicaciones distribuidas. Para mejorar el ciclo de desarrollo interno de las aplicaciones preparadas para la nube, se han creado integraciones con .NET Aspire para conectar las aplicaciones .NET a los recursos AWS. Estas integraciones están disponibles a través del paquete [NuGet Aspire.Hosting.aws](#).

Están disponibles las siguientes integraciones de .NET Aspire:

- La capacidad de aprovisionar sus AWS recursos mediante [CloudFormation](#). Esta integración se utiliza en el proyecto .NET Aspire AppHost .

Para obtener más información, consulte la entrada del blog [Integración AWS con .NET Aspire](#).

- Instalación, configuración y conexión AWS SDK para .NET a [Amazon DynamoDB local](#). Esta integración se utiliza en el proyecto .NET Aspire AppHost .

Para obtener más información, consulte la entrada del blog [Integración AWS con .NET Aspire](#).

- Habilite un entorno de desarrollo local para [AWS Lambda](#) las funciones. Esta integración se utiliza en el proyecto.NET Aspire AppHost .

Para obtener más información, consulte las entradas del blog [Creación y depuración de aplicaciones de.NET Lambda con .NET Aspire \(parte 1\)](#) y [Creación y depuración de aplicaciones de.NET Lambda con .NET Aspire \(parte 2\)](#).

Note

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

Como esta función está en versión preliminar, tendrá que optar por activarla. Para obtener más información sobre esta función de vista previa y sobre cómo activarla, consulta el [tema del seguimiento de desarrollos](#) en GitHub

Información adicional

Para obtener información adicional y detalles sobre cómo puede utilizar las integraciones disponibles en [Aspire.Hosting.aws](#), consulte los siguientes recursos.

- [Entrada de blog Integración con.NET Aspire. AWS](#)
- Publicaciones de blog [Creación y depuración de aplicaciones .NET Lambda con .NET Aspire \(parte 1\)](#) y [creación y depuración de aplicaciones de.NET Lambda con .NET Aspire \(parte 2\)](#).
- [El integrations-on-dotnet-aspire repositorio -for-aws está disponible](#). GitHub
- [El archivo README detallado del paquete Aspire.Hosting.aws](#). NuGet

Support para otros AWS servicios y configuraciones

Los AWS servicios de AWS SDK para .NET soporte se suman a los descritos en las secciones anteriores. Para obtener información sobre todos los servicios compatibles, consulte la [referencia de la AWS SDK para .NET API](#). APIs

Además de los espacios de nombres para los AWS servicios individuales, AWS SDK para .NET también proporciona lo siguiente: APIs

Área	Descripción	Recursos
AWS Support	Acceso programático a los casos de AWS Support y a las funciones de Trusted Advisor.	Consulte Amazon.AWSSupport y Amazon.AWSSupport.Modelo .
General	Clases auxiliares y enumeraciones.	Consulte Amazon y Amazon.Util .

SDK para .NET ejemplos de código (v4)

Los ejemplos de código de este tema muestran cómo usar la AWS SDK para .NET (v4) con AWS.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

Algunos servicios contienen categorías de ejemplo adicionales que muestran cómo aprovechar las bibliotecas o funciones específicas del servicio.

Services

- [Ejemplos de Aurora utilizando SDK para .NET \(v4\)](#)
- [Ejemplos de Auto Scaling usando SDK para .NET \(v4\)](#)
- [Ejemplos de Amazon Bedrock que utilizan SDK para .NET \(v4\)](#)
- [Ejemplos de Amazon Bedrock Runtime con SDK para .NET \(v4\)](#)
- [CloudFormation ejemplos que utilizan SDK para .NET \(v4\)](#)
- [CloudWatch ejemplos que utilizan SDK para .NET \(v4\)](#)
- [CloudWatch Registra ejemplos con SDK para .NET \(v4\)](#)
- [Ejemplos de proveedores de identidad de Amazon Cognito con SDK para .NET \(v4\)](#)
- [AWS Control Tower ejemplos que utilizan SDK para .NET \(v4\)](#)
- [Ejemplos SDK para .NET de DynamoDB con \(v4\)](#)
- [Ejemplos de Amazon EC2 con SDK para .NET \(v4\)](#)
- [Ejemplos de Amazon ECS con SDK para .NET \(v4\)](#)
- [AWS IoT ejemplos que utilizan SDK para .NET \(v4\)](#)
- [AWS IoT data ejemplos que utilizan SDK para .NET \(v4\)](#)
- [Ejemplos de Amazon Redshift con SDK para .NET \(v4\)](#)
- [Ejemplos de Amazon S3 con SDK para .NET \(v4\)](#)

Ejemplos de Aurora utilizando SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con Aurora.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

Introducción a Aurora

En el siguiente ejemplo de código, se muestra cómo empezar a utilizar Aurora.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.RDS;  
using Amazon.RDS.Model;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here for
example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\\tCluster: database: {cluster.DatabaseName} identifier:
{cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Para obtener más información sobre la API, consulta la [sección Describir DBClusters](#) en la referencia de la AWS SDK para .NET API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Cree un grupo de parámetros de clúster de base de datos de Aurora y defina los valores de los parámetros.
- Cree un clúster de base de datos que utilice el grupo de parámetros.
- Cree una instancia de base de datos que contenga una base de datos.
- Realice una instantánea del clúster de base de datos y luego limpie los recursos.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
```

```

/// </summary>
public class AuroraScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        This .NET example performs the following tasks:
        1. Return a list of the available DB engine families for Aurora MySQL using the
        DescribeDBEngineVersionsAsync method.
        2. Select an engine family and create a custom DB cluster parameter group using
        the CreateDBClusterParameterGroupAsync method.
        3. Get the parameter group using the DescribeDBClusterParameterGroupsAsync
        method.
        4. Get some parameters in the group using the DescribeDBClusterParametersAsync
        method.
        5. Parse and display some parameters in the group.
        6. Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
        7. Get and display the updated parameters using the
        DescribeDBClusterParametersAsync method with a source of "user".
        8. Get a list of allowed engine versions using the
        DescribeDBEngineVersionsAsync method.
        9. Create an Aurora DB cluster that contains a MySQL database and uses the
        parameter group.
            using the CreateDBClusterAsync method.
        10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
        method.
        11. Display and select from a list of instance classes available for the
        selected engine and version
            using the paginated DescribeOrderableDBInstanceOptions method.
        12. Create a database instance in the cluster using the CreateDBInstanceAsync
        method.
        13. Wait for the DB instance to be ready using the DescribeDBInstances method.
        14. Display the connection endpoint string for the new DB cluster.
        15. Create a snapshot of the DB cluster using the CreateDBClusterSnapshotAsync
        method.
        16. Wait for DB snapshot to be ready using the DescribeDBClusterSnapshotsAsync
        method.
        17. Delete the DB instance using the DeleteDBInstanceAsync method.
        18. Delete the DB cluster using the DeleteDBClusterAsync method.
        19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync methods.
        20. Delete the cluster parameter group using the
        DeleteDBClusterParameterGroupAsync.
    */
}

```



```

*/

private static readonly string sepBar = new('-', 80);
private static AuroraWrapper auroraWrapper = null!;
private static ILogger logger = null!;
private static readonly string engine = "aurora-mysql";
static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon Relational Database Service
    (Amazon RDS).
    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<AuroraWrapper>()
        )
        .Build();

    logger = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
    }).CreateLogger<AuroraScenario>();

    auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

    Console.WriteLine(sepBar);
    Console.WriteLine(
        "Welcome to the Amazon Aurora: get started with DB clusters example.");
    Console.WriteLine(sepBar);

    DBClusterParameterGroup parameterGroup = null!;
    DBCluster? newCluster = null;
    DBInstance? newInstance = null;

    try
    {
        var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

        parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

```

```
        var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

        await ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName,
parameters);

        await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

        var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

        var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

        newCluster = await CreateNewCluster
        (
            parameterGroup,
            engine,
            engineVersionChoice.EngineVersion,
            newClusterIdentifier
        );

        var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

        var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

        newInstance = await CreateNewInstance(
            newClusterIdentifier,
            engine,
            engineVersionChoice.EngineVersion,
            instanceClassChoice.DBInstanceClass,
            newInstanceIdentifier
        );

        DisplayConnectionString(newCluster!);
        await CreateSnapshot(newCluster!);
        await CleanupResources(newInstance, newCluster, parameterGroup);

        Console.WriteLine("Scenario complete.");
        Console.WriteLine(sepBar);
```

```

    }
    catch (Exception ex)
    {
        await CleanupResources(newInstance, newCluster, parameterGroup);
        logger.LogError(ex, "There was a problem executing the scenario.");
    }
}

/// <summary>
/// Choose the Aurora DB parameter group family from a list of available
options.
/// </summary>
/// <returns>The selected parameter group family.</returns>
public static async Task<string> ChooseParameterGroupFamilyAsync()
{
    Console.WriteLine(sepBar);
    // 1. Get a list of available engines.
    var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

    Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

    var parameterGroupFamilies =
        engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
    for (var i = 1; i <= parameterGroupFamilies.Count; i++)
    {
        var parameterGroupFamily = parameterGroupFamilies[i - 1];
        // List the available parameter group families.
        Console.WriteLine(
            $"{i}. Family: {parameterGroupFamily.Key}");
    }

    var choiceNumber = 0;
    while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
    {
        Console.WriteLine("2. Select an available DB parameter group family by
entering a number from the preceding list:");
        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<DBClusterParameterGroup>
CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}");

        var parameterGroup = await
auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParameterG

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The 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>> DescribeParametersInGroupAsync(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
    }

```

```

        Console.WriteLine(sepBar);

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\\n\\tParameter: {p.ParameterName}." +
                $"\\n\\tDescription: {p.Description}." +
                $"\\n\\tAllowed Values: {p.AllowedValues}." +
                $"\\n\\tValue: {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 ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>

```

```

    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName, "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"{p.ParameterName}." +
                $"{p.Description}." +
                $"{p.AllowedValues}." +
                $"{p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"{i}. {version.DBEngineVersionDescription}");

```

```

        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>
/// Create a new RDS DB cluster.
/// </summary>
/// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
/// <param name="engineName">Engine to use for the DB cluster.</param>
/// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
/// <param name="clusterIdentifier">Cluster identifier to use for the DB
cluster.</param>
/// <returns>The new DB cluster.</returns>
public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
parameterGroup,
    string engineName, string engineVersion, string clusterIdentifier)
{
    Console.WriteLine(sepBar);
    Console.WriteLine($"9. Create a new DB cluster with identifier
{clusterIdentifier}.");

    DBCluster newCluster;
    var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
    var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
clusterIdentifier);

    if (isClusterCreated)
    {
        Console.WriteLine("Cluster already created.");
    }
}

```

```

        newCluster = clusters.First(i => i.DBClusterIdentifier ==
clusterIdentifier);
    }
    else
    {
        Console.WriteLine("Enter an admin username:");
        var username = Console.ReadLine();

        Console.WriteLine("Enter an admin password:");
        var password = Console.ReadLine();

        newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
            "ExampleDatabase",
            clusterIdentifier,
            parameterGroup.DBClusterParameterGroupName,
            engineName,
            engineVersion,
            username!,
            password!
        );

        Console.WriteLine("10. Waiting for DB cluster to be ready...");
        while (newCluster.Status != "available")
        {
            Console.Write(".");
            Thread.Sleep(5000);
            clusters = await
auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
            newCluster = clusters.First();
        }
    }

    Console.WriteLine(sepBar);
    return newCluster;
}

/// <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 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\\t{i}. Instance class: {instance.DBInstanceClass} (storage type
{instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <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?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {
            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);

```

```

        isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
        newInstance = instances.First();
    }
}

Console.WriteLine(sepBar);
return newInstance;
}

/// <summary>
/// Display a connection string for an Amazon RDS DB cluster.
/// </summary>
/// <param name="cluster">The DB cluster to use to get a connection string.</
param>
public static void DisplayConnectionString(DBCluster cluster)
{
    Console.WriteLine(sepBar);
    // Display the connection string.
    Console.WriteLine("14. New DB cluster connection string: ");
    Console.WriteLine(
        $"\\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
        + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\\n");

    Console.WriteLine(sepBar);
}

/// <summary>
/// Create a snapshot from an Amazon RDS DB cluster.
/// </summary>
/// <param name="cluster">DB cluster to use when creating a snapshot.</param>
/// <returns>The snapshot object.</returns>
public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
{
    Console.WriteLine(sepBar);
    // Create a snapshot.
    Console.WriteLine($"15. Creating snapshot from DB cluster
{cluster.DBClusterIdentifier}.");
    var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
        cluster.DBClusterIdentifier,
        "ExampleSnapshot-" + DateTime.Now.Ticks);

    // Wait for the snapshot to be available.
    bool isSnapshotReady = false;

```

```

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }
    }

```

```

        if (newCluster is not null && GetYesNoResponse($"\\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// 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;
}
```

Métodos de contenedor que llama el escenario para administrar las acciones de Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
    string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
```

```

        {
            Engine = engine,
            DBParameterGroupFamily = parameterGroupFamily
        });
    return response.DBEngineVersions;
}

/// <summary>
/// Create a custom cluster parameter group.
/// </summary>
/// <param name="parameterGroupFamily">The family of the parameter group.</
param>
/// <param name="groupName">The name for the new parameter group.</param>
/// <param name="description">A description for the new parameter group.</param>
/// <returns>The new parameter group object.</returns>
public async Task<DBClusterParameterGroup>
CreateCustomClusterParameterGroupAsync(
    string parameterGroupFamily,
    string groupName,
    string description)
{
    var request = new CreateDBClusterParameterGroupRequest
    {
        DBParameterGroupFamily = parameterGroupFamily,
        DBClusterParameterGroupName = groupName,
        Description = description,
    };

    var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
    return response.DBClusterParameterGroup;
}

/// <summary>
/// Describe the cluster parameters in a parameter group.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <param name="source">The optional name of the source filter.</param>
/// <returns>The collection of parameters.</returns>
public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
{
    var paramList = new List<Parameter>();

    DescribeDBClusterParametersResponse response;

```

```

        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)

```



```

    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }

    /// <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>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
        _amazonRDS.Paginateors.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>
/// Delete a particular parameter group by name.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterParameterGroupNameAsync(string groupName)
{
    var request = new DeleteDBClusterParameterGroupRequest
    {
        DBClusterParameterGroupName = groupName,
    };

    var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,

```

```

        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <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>> DescribeDBInstancesPagedAsync(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>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon 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="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</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>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,

```

```

        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>

```

```

    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceId">DB instance identifier.</param>

```

```
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdIdentifier = dbInstanceIdIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para .NET .

- [CrearDBCluster](#)
- [CrearDBClusterParameterGroup](#)
- [Crear DBCluster instantánea](#)
- [CrearDBInstance](#)
- [DeleteDBCluster](#)
- [DeleteDBClusterParameterGroup](#)
- [DeleteDBInstance](#)
- [DescribirDBClusterParameterGroups](#)
- [Describa DBCluster los parámetros](#)
- [Describa las DBCluster instantáneas](#)
- [DescribirDBClusters](#)
- [Describa las versiones DBEngine](#)
- [DescribirDBInstances](#)
- [DescribeOrderableDBInstanceOpciones](#)
- [ModifyDBClusterParameterGroup](#)

Acciones

CreateDBCluster

En el siguiente ejemplo de código, se muestra cómo utilizar CreateDBCluster.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        MasterUsername = adminName,
```



```

        MasterUserPassword = adminPassword,
    };

    var response = await _amazonRDS.CreateDBClusterAsync(request);
    return response.DBCluster;
}

```

- Para obtener más información sobre la API, consulta la [sección Crear DBCluster](#) en la referencia de la AWS SDK para .NET API.

CreateDBClusterParameterGroup

En el siguiente ejemplo de código, se muestra cómo utilizar CreateDBClusterParameterGroup.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Create a custom cluster parameter group.
/// </summary>
/// <param name="parameterGroupFamily">The family of the parameter group.</
param>
/// <param name="groupName">The name for the new parameter group.</param>
/// <param name="description">A description for the new parameter group.</param>
/// <returns>The new parameter group object.</returns>
public async Task<DBClusterParameterGroup>
CreateCustomClusterParameterGroupAsync(
    string parameterGroupFamily,
    string groupName,
    string description)
{
    var request = new CreateDBClusterParameterGroupRequest
    {
        DBParameterGroupFamily = parameterGroupFamily,
        DBClusterParameterGroupName = groupName,
    }
}

```

```

        Description = description,
    };

    var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
    return response.DBClusterParameterGroup;
}

```

- Para obtener más información sobre la API, consulta la [sección Crear DBClusterParameterGroup](#) en la referencia de la AWS SDK para .NET API.

CreateDBClusterSnapshot

En el siguiente ejemplo de código, se muestra cómo utilizar CreateDBClusterSnapshot.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Create a snapshot of a cluster.
/// </summary>
/// <param name="dbClusterIdentifier">DB cluster identifier.</param>
/// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
/// <returns>DB snapshot object.</returns>
public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
{
    var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
        new CreateDBClusterSnapshotRequest()
        {
            DBClusterIdentifier = dbClusterIdentifier,
            DBClusterSnapshotIdentifier = snapshotIdentifier,
        });

    return response.DBClusterSnapshot;
}

```

```
}
```

- Para obtener más información sobre la API, consulta [Crear DBCluster una instantánea](#) en la referencia AWS SDK para .NET de la API.

CreateDBInstance

En el siguiente ejemplo de código, se muestra cómo utilizar CreateDBInstance.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Create an Amazon Relational Database Service (Amazon 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="dbInstanceId">DB instance identifier.</param>
/// <param name="dbClusterIdentifier">DB cluster identifier.</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>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> CreateDBInstanceInClusterAsync(
    string dbClusterIdentifier,
    string dbInstanceId,
    string dbEngine,
    string dbEngineVersion,
    string instanceClass)
{
    // When creating the instance within a cluster, do not specify the name or
    size.
    var response = await _amazonRDS.CreateDBInstanceAsync(
        new CreateDBInstanceRequest()
        {
```

```

        DBClusterIdentifier = dbClusterIdentifier,
        DBInstanceIdentifier = dbInstanceIdentifier,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        DBInstanceClass = instanceClass
    });

    return response.DBInstance;
}

```

- Para obtener más información sobre la API, consulta la [sección Crear DBInstance](#) en la referencia de la AWS SDK para .NET API.

DeleteDBCluster

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteDBCluster.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Delete a particular DB cluster.
/// </summary>
/// <param name="dbClusterIdentifier">DB cluster identifier.</param>
/// <returns>DB cluster object.</returns>
public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
{
    var response = await _amazonRDS.DeleteDBClusterAsync(
        new DeleteDBClusterRequest()
        {
            DBClusterIdentifier = dbClusterIdentifier,
            SkipFinalSnapshot = true
        });
}

```

```
        return response.DBCluster;
    }
```

- Para obtener más información sobre la API, consulta [Eliminar DBCluster](#) en la referencia AWS SDK para .NET de la API.

DeleteDBClusterParameterGroup

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteDBClusterParameterGroup.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete a particular parameter group by name.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
{
    var request = new DeleteDBClusterParameterGroupRequest
    {
        DBClusterParameterGroupName = groupName,
    };

    var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [Eliminar DBCluster ParameterGroup](#) en la referencia AWS SDK para .NET de la API.

DeleteDBInstance

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteDBInstance.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}
```

- Para obtener más información sobre la API, consulta [Eliminar DBInstance](#) en la referencia AWS SDK para .NET de la API.

DescribeDBClusterParameterGroups

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeDBClusterParameterGroups.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get the description of a DB cluster parameter group by name.
/// </summary>
/// <param name="name">The name of the DB parameter group to describe.</param>
/// <returns>The parameter group description.</returns>
public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
{
    var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
        new DescribeDBClusterParameterGroupsRequest()
        {
            DBClusterParameterGroupName = name
        });
    return response.DBClusterParameterGroups.FirstOrDefault();
}
```

- Para obtener más información sobre la API, consulta la [sección Describir DBClusterParameterGroups](#) en la referencia de la AWS SDK para .NET API.

DescribeDBClusterParameters

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeDBClusterParameters`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

```

- Para obtener información detallada sobre la API, consulta [Describir DBCluster los parámetros](#) en la referencia de la AWS SDK para .NET API.

DescribeDBClusterSnapshots

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeDBClusterSnapshots.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Return a list of DB snapshots for a particular DB cluster.
/// </summary>
/// <param name="dbClusterIdentifier">DB cluster identifier.</param>
/// <returns>List of DB snapshots.</returns>
public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
{
    var results = new List<DBClusterSnapshot>();

    DescribeDBClusterSnapshotsResponse response;
    DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
    {
        DBClusterIdentifier = dbClusterIdentifier
    };
    // Get the full list if there are multiple pages.
    do
    {
        response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
        results.AddRange(response.DBClusterSnapshots);
        request.Marker = response.Marker;
    }
    while (response.Marker is not null);
    return results;
}
```

- Para obtener más información sobre la API, consulte [Describir DBCluster las instantáneas](#) en la referencia de AWS SDK para .NET la API.

DescribeDBClusters

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeDBClusters.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Returns a list of DB clusters.
/// </summary>
/// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
/// <returns>List of DB clusters.</returns>
public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
{
    var results = new List<DBCluster>();

    DescribeDBClustersResponse response;
    DescribeDBClustersRequest request = new DescribeDBClustersRequest
    {
        DBClusterIdentifier = dbClusterIdentifier
    };
    // Get the full list if there are multiple pages.
    do
    {
        response = await _amazonRDS.DescribeDBClustersAsync(request);
        if (response.DBClusters != null)
        {
            results.AddRange(response.DBClusters);
        }
        request.Marker = response.Marker;
    }
    while (response.Marker is not null);
    return results;
}
```

- Para obtener más información sobre la API, consulta la [sección Describir DBClusters](#) en la referencia de la AWS SDK para .NET API.

DescribeDBEngineVersions

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeDBEngineVersions.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get a list of DB engine versions for a particular DB engine.
/// </summary>
/// <param name="engine">The name of the engine.</param>
/// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
/// <returns>A list of DBEngineVersions.</returns>
public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
    string? parameterGroupFamily = null)
{
    var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
        new DescribeDBEngineVersionsRequest()
        {
            Engine = engine,
            DBParameterGroupFamily = parameterGroupFamily
        });
    return response.DBEngineVersions;
}
```

- Para obtener más información sobre la API, consulta la [sección Describir DBEngine las versiones](#) en la referencia de la AWS SDK para .NET API.

DescribeDBInstances

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeDBInstances.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Returns a list of DB instances.
/// </summary>
/// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
/// <returns>List of DB instances.</returns>
public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(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;
}
```

- Para obtener más información sobre la API, consulta la [sección Describir DBInstances](#) en la referencia de la AWS SDK para .NET API.

DescribeOrderableDBInstanceOptions

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeOrderableDBInstanceOptions`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get a list of orderable DB instance options for a specific
/// engine and engine version.
/// </summary>
/// <param name="engine">Name of the engine.</param>
/// <param name="engineVersion">Version of the engine.</param>
/// <returns>List of OrderableDBInstanceOptions.</returns>
public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(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 obtener más información sobre la API, consulta [DescribeOrderableDBInstances opciones](#) en la referencia AWS SDK para .NET de la API.

ModifyDBClusterParameterGroup

En el siguiente ejemplo de código, se muestra cómo utilizar ModifyDBClusterParameterGroup.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Modify the specified integer parameters with new values from user input.
/// </summary>
/// <param name="groupName">The group name for the parameters.</param>
/// <param name="parameters">The list of integer parameters to modify.</param>
/// <param name="newValue">Optional int value to set for parameters.</param>
/// <returns>The name of the group that was modified.</returns>
public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
{
    foreach (var p in parameters)
    {
        if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
        {
            while (newValue == 0)
            {
                Console.WriteLine(
                    $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                var choice = Console.ReadLine();
                int.TryParse(choice, out newValue);
            }

            p.ParameterValue = newValue.ToString();
        }
    }
}

```

```
var request = new ModifyDBClusterParameterGroupRequest
{
    Parameters = parameters,
    DBClusterParameterGroupName = groupName,
};

var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
return result.DBClusterParameterGroupName;
}
```

- Para obtener más información sobre la API, consulta la [sección Modificar DBCluster ParameterGroup](#) en la referencia de la AWS SDK para .NET API.

Ejemplos de Auto Scaling usando SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con Auto Scaling.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

Introducción al escalado automático

El siguiente ejemplo de código muestra cómo empezar a usar Auto Scaling.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
        Console.WriteLine("Let's get a description of your Auto Scaling groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

        if (response.AutoScalingGroups == null || response.AutoScalingGroups.Count
== 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
groups.");
            return;
        }
    }
}
```



```
        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {
            Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.AvailabilityZone}");
        });
    }
}
```

- Para obtener más información sobre la API, consulta [DescribeAutoScalingGroups](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crear un grupo de Amazon EC2 Auto Scaling con una plantilla de lanzamiento y zonas de disponibilidad y obtener información sobre las instancias en ejecución
- Habilita la recopilación de CloudWatch métricas de Amazon.
- Actualizar la capacidad deseada del grupo y esperar a que una instancia se inicie
- Terminar una instancia del grupo.
- Mostrar las actividades de escalado que se producen como respuesta a las solicitudes de los usuarios y a los cambios de capacidad
- Obtén estadísticas para CloudWatch las métricas y, a continuación, limpia los recursos.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
global using Amazon.AutoScaling;
```

```
global using Amazon.AutoScaling.Model;
global using Amazon.CloudWatch;
global using AutoScalingActions;
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.EC2;
using Microsoft.Extensions.Configuration;
using Host = Microsoft.Extensions.Hosting.Host;

namespace AutoScalingBasics;

public class AutoScalingBasics
{
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 Auto Scaling, Amazon
        // CloudWatch, and Amazon EC2.
        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<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonCloudWatch>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalingWrapper>()
                    .AddTransient<CloudWatchWrapper>()
                    .AddTransient<EC2Wrapper>()
                    .AddTransient<UIWrapper>()
                )
            .Build();

        var autoScalingWrapper =
            host.Services.GetRequiredService<AutoScalingWrapper>();
    }
}
```

```

    var cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();
    var ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
    var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

    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 imageId = configuration["ImageId"];
    var instanceType = configuration["InstanceType"];
    var launchTemplateName = configuration["LaunchTemplateName"];

    launchTemplateName += Guid.NewGuid().ToString();

    // The name of the Auto Scaling group.
    var groupName = configuration["GroupName"];

    uiWrapper.DisplayTitle("Auto Scaling Basics");
    uiWrapper.DisplayAutoScalingBasicsDescription();

    // Create the launch template and save the template Id to use when deleting
the
    // launch template at the end of the application.
    var launchTemplateId = await ec2Wrapper.CreateLaunchTemplateAsync(imageId!,
instanceType!, launchTemplateName);

    // Confirm that the template was created by asking for a description of it.
    await ec2Wrapper.DescribeLaunchTemplateAsync(launchTemplateName);

    uiWrapper.PressEnter();

    var availabilityZones = await ec2Wrapper.ListAvailabilityZonesAsync();

    Console.WriteLine($"Creating an Auto Scaling group named {groupName}.");
    await autoScalingWrapper.CreateAutoScalingGroupAsync(
        groupName!,
        launchTemplateName,
        availabilityZones[0].ZoneName);

    // Keep checking the details of the new group until its lifecycle state

```

```
// is "InService".
Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

List<AutoScalingInstanceDetails> instanceDetails;

do
{
    instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
}
while (instanceDetails.Count <= 0);

Console.WriteLine($"Auto scaling group {groupName} successfully created.");
Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

// Display the details of the Auto Scaling group.
instanceDetails.ForEach(detail =>
{
    Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
});

uiWrapper.PressEnter();

uiWrapper.DisplayTitle("Metrics collection");
Console.WriteLine($"Enable metrics collection for {groupName}");
await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

// Show the metrics that are collected for the group.

// Update the maximum size of the group to three instances.
Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
int maxSize = 3;
await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

uiWrapper.DisplayGroupDetails(groups!);
```

```

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {

```

```
        Console.WriteLine($"The activity Id is {activity.ActivityId}");
        Console.WriteLine($"The activity details are {activity.Details}");
    });
}

// Display the Amazon CloudWatch metrics that have been collected.
var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
if (metrics.Any())
{
    Console.WriteLine($"Metrics collected for {groupName}:");
    metrics.ForEach(metric =>
    {
        Console.Write($"Metric name: {metric.MetricName}\t");
        Console.WriteLine($"Namespace: {metric.Namespace}");
    });
}

var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
if (dataPoints.Any())
{
    Console.WriteLine("Details for the metrics collected:");
    dataPoints.ForEach(detail => { Console.WriteLine(detail); });
}

// Disable metrics collection.
Console.WriteLine("Disabling the collection of metrics for {groupName}.");
var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

if (success)
{
    Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
}
else
{
    Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
}

// Terminate all instances in the group.
uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
```

```

        Console.WriteLine("Now terminating all instances in the Auto Scaling
group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}

namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>

```

```

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;
    }
}

```



```

        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <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();
    }
}

```

Definir las funciones a las que llama el escenario para administrar plantillas de lanzamiento y métricas. Estas funciones incluyen Auto Scaling, Amazon EC2 y CloudWatch acciones.

```

namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
    client.</param>

```

```

public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
{
    _amazonAutoScaling = amazonAutoScaling;
}

/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
    {
        availabilityZone,
    };

    var request = new CreateAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        AvailabilityZones = zoneList,
        LaunchTemplate = templateSpecification,
        MaxSize = 6,
        MinSize = 1
    };
    try
    {
        var response = await
            _amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

```

        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }

    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
            response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
            response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };
    }

```

```

        var response = await
_amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }

    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
DescribeAutoScalingInstancesAsync(
    string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

```

```

        var response =
            await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                scalingGroupsRequest);
        if (response.AutoScalingInstances != null)
        {
            instanceDetails = response.AutoScalingInstances;
        }
    }

    return instanceDetails;
}

/// <summary>
/// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
/// </summary>
/// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
/// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
    string groupName)
{
    var groups = new List<AutoScalingGroup>();
    var groupList = new List<string>
    {
        groupName,
    };

    var request = new DescribeAutoScalingGroupsRequest
    {
        AutoScalingGroupNames = groupList,
    };

    var response = await
        _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
    if (response.AutoScalingGroups != null)
    {
        groups = response.AutoScalingGroups;
    }

    return groups;
}

```

```

    /// <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;
    }

    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };
    }

```

```

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
        {
            "GroupMaxSize",
        };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {

```



```

        AutoScalingGroupName = groupName,
        DesiredCapacity = desiredCapacity,
    };

    var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
    Console.WriteLine($"You have set the DesiredCapacity to
{desiredCapacity}.");

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Terminate all instances in the Auto Scaling group in preparation for
/// deleting the group.
/// </summary>
/// <param name="instanceId">The instance Id of the instance to terminate.</
param>
/// <returns>A Boolean value that indicates the success or failure of
/// the operation.</returns>
public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
    string instanceId)
{
    var request = new TerminateInstanceInAutoScalingGroupRequest
    {
        InstanceId = instanceId,
        ShouldDecrementDesiredCapacity = false,
    };

    var response = await
_amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"You have terminated the instance: {instanceId}");
        return true;
    }

    Console.WriteLine($"Could not terminate {instanceId}");
    return false;
}

```

```

    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
        _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
{groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }
}

namespace AutoScalingActions;

```

```
using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }
}
```

```

    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }
    }

```

```

        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}

namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>

```

```

/// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
{
    var metrics = new List<Metric>();
    var filter = new DimensionFilter
    {
        Name = "AutoScalingGroupName",
        Value = $"{groupName}",
    };

    var request = new ListMetricsRequest
    {
        MetricName = "AutoScalingGroupName",
        Dimensions = new List<DimensionFilter> { filter },
        Namespace = "AWS/AutoScaling",
    };

    var response = await _amazonCloudWatch.ListMetricsAsync(request);
    if (response.Metrics != null)
    {
        metrics = response.Metrics;
    }
    return metrics;
}

/// <summary>
/// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
/// <returns>A list of data points.</returns>
public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
{
    var dataPoints = new List<Datapoint>();
    var metricDimensions = new List<Dimension>
    {
        new Dimension
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        },
    };

    // The start time will be yesterday.

```

```
var startTime = DateTime.UtcNow.AddDays(-1);

var request = new GetMetricStatisticsRequest
{
    MetricName = "AutoScalingGroupName",
    Dimensions = metricDimensions,
    Namespace = "AWS/AutoScaling",
    Period = 60, // 60 seconds.
    Statistics = new List<string>() { "Minimum" },
    StartTimeUtc = startTime,
    EndTimeUtc = DateTime.UtcNow,
};

var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
if (response.Datapoints != null)
{
    dataPoints = response.Datapoints;
}

return dataPoints;
}

}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAutoScalingInstances](#)
 - [DescribeScalingActivities](#)
 - [DisableMetricsCollection](#)
 - [EnableMetricsCollection](#)
 - [SetDesiredCapacity](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Acciones

CreateAutoScalingGroup

En el siguiente ejemplo de código, se muestra cómo utilizar `CreateAutoScalingGroup`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
    {
        availabilityZone,
    };

    var request = new CreateAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        AvailabilityZones = zoneList,
    };
}
```



```

        LaunchTemplate = templateSpecification,
        MaxSize = 6,
        MinSize = 1
    };
    try
    {
        var response = await
        _amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AlreadyExistsException)
    {
        Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
        return true;
    }
}

```

- Para obtener más información sobre la API, consulta [CreateAutoScalingGroup](#) la Referencia AWS SDK para .NET de la API.

DescribeAutoScalingGroups

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeAutoScalingGroups`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Get data about the instances in an Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>

```

```
/// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
public async Task<List<AutoScalingInstanceDetails>>
DescribeAutoScalingInstancesAsync(
    string groupName)
{
    var groups = await DescribeAutoScalingGroupsAsync(groupName);
    var instanceIds = new List<string>();
    var instanceDetails = new List<AutoScalingInstanceDetails>();
    if (groups != null)
    {
        groups.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response =
            await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                scalingGroupsRequest);
        if (response.AutoScalingInstances != null)
        {
            instanceDetails = response.AutoScalingInstances;
        }
    }

    return instanceDetails;
}
```

- Para obtener más información sobre la API, consulta [DescribeAutoScalingGroups](#) la Referencia AWS SDK para .NET de la API.

DescribeAutoScalingInstances

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeAutoScalingInstances.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
DescribeAutoScalingInstancesAsync(
    string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });
        }
    }

```

```
var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
{
    MaxRecords = 10,
    InstanceIds = instanceIds,
};

var response =
    await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
        scalingGroupsRequest);
if (response.AutoScalingInstances != null)
{
    instanceDetails = response.AutoScalingInstances;
}

return instanceDetails;
}
```

- Para obtener más información sobre la API, consulta [DescribeAutoScalingInstances](#) la Referencia AWS SDK para .NET de la API.

DescribeScalingActivities

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeScalingActivities`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
/// Amazon EC2 Auto Scaling group.
/// </summary>
```

```

    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
        _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }

```

- Para obtener más información sobre la API, consulta [DescribeScalingActivities](#) la Referencia AWS SDK para .NET de la API.

DisableMetricsCollection

En el siguiente ejemplo de código, se muestra cómo utilizar `DisableMetricsCollection`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling

```

```
/// group.  
/// </summary>  
/// <param name="groupName">The name of the Auto Scaling group.</param>  
/// <returns>A Boolean value that indicates the success or failure of  
/// the operation.</returns>  
public async Task<bool> DisableMetricsCollectionAsync(string groupName)  
{  
    var request = new DisableMetricsCollectionRequest  
    {  
        AutoScalingGroupName = groupName,  
    };  
  
    var response = await  
_amazonAutoScaling.DisableMetricsCollectionAsync(request);  
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
}
```

- Para obtener más información sobre la API, consulta [DisableMetricsCollection](#) la Referencia AWS SDK para .NET de la API.

EnableMetricsCollection

En el siguiente ejemplo de código, se muestra cómo utilizar `EnableMetricsCollection`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// Enable the collection of metric data for an Auto Scaling group.  
/// </summary>  
/// <param name="groupName">The name of the Auto Scaling group.</param>  
/// <returns>A Boolean value indicating the success of the action.</returns>  
public async Task<bool> EnableMetricsCollectionAsync(string groupName)  
{
```

```

var listMetrics = new List<string>
{
    "GroupMaxSize",
};

var collectionRequest = new EnableMetricsCollectionRequest
{
    AutoScalingGroupName = groupName,
    Metrics = listMetrics,
    Granularity = "1Minute",
};

var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- Para obtener más información sobre la API, consulta [EnableMetricsCollection](#) la Referencia AWS SDK para .NET de la API.

SetDesiredCapacity

En el siguiente ejemplo de código, se muestra cómo utilizar SetDesiredCapacity.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Set the desired capacity of an Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Auto Scaling group.</param>
/// <param name="desiredCapacity">The desired capacity for the Auto
/// Scaling group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>

```

```
public async Task<bool> SetDesiredCapacityAsync(
    string groupName,
    int desiredCapacity)
{
    var capacityRequest = new SetDesiredCapacityRequest
    {
        AutoScalingGroupName = groupName,
        DesiredCapacity = desiredCapacity,
    };

    var response = await
        _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
    Console.WriteLine($"You have set the DesiredCapacity to
        {desiredCapacity}.");

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [SetDesiredCapacity](#) la Referencia AWS SDK para .NET de la API.

TerminateInstanceInAutoScalingGroup

En el siguiente ejemplo de código, se muestra cómo utilizar `TerminateInstanceInAutoScalingGroup`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Terminate all instances in the Auto Scaling group in preparation for
/// deleting the group.
/// </summary>
```



```
/// <param name="instanceId">The instance Id of the instance to terminate.</param>
/// <returns>A Boolean value that indicates the success or failure of
/// the operation.</returns>
public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
    string instanceId)
{
    var request = new TerminateInstanceInAutoScalingGroupRequest
    {
        InstanceId = instanceId,
        ShouldDecrementDesiredCapacity = false,
    };

    var response = await
        _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"You have terminated the instance: {instanceId}");
        return true;
    }

    Console.WriteLine($"Could not terminate {instanceId}");
    return false;
}
```

- Para obtener más información sobre la API, consulta [TerminateInstanceInAutoScalingGroup](#) la Referencia AWS SDK para .NET de la API.

UpdateAutoScalingGroup

En el siguiente ejemplo de código, se muestra cómo utilizar UpdateAutoScalingGroup.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
        _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
{groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [UpdateAutoScalingGroup](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon Bedrock que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon Bedrock.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Acciones](#)

Introducción

Introducción a Amazon Bedrock

En el siguiente ejemplo de código se muestra cómo empezar a utilizar Amazon Bedrock.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace BedrockActions;
```

```
/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
        // of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);
    }

    /// <summary>
    /// List foundation models.
    /// </summary>
    /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
    private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
    {
        Console.WriteLine("List foundation models with no filter.");

        try
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
        }
    }
}
```

```

        else
        {
            Console.WriteLine("Something wrong happened");
        }
    }
    catch (AmazonBedrockException e)
    {
        Console.WriteLine(e.Message);
    }
}

/// <summary>
/// Write the foundation model summary to console.
/// </summary>
/// <param name="foundationModel"> The foundation model summary to write to
console. </param>
private static void WriteToConsole(FoundationModelSummary foundationModel)
{
    Console.WriteLine($"{foundationModel.ModelId}, Customization:
{string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
{foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
", foundationModel.InputModalities)}, Output: {string.Join(", ",
foundationModel.OutputModalities)}}");
}
}

```

- Para obtener más información sobre la API, consulta [ListFoundationModels](#) la Referencia AWS SDK para .NET de la API.

Acciones

ListFoundationModels

En el siguiente ejemplo de código, se muestra cómo utilizar `ListFoundationModels`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Enumerar los modelos fundacionales Bedrock disponibles

```
/// <summary>
/// List foundation models.
/// </summary>
/// <param name="bedrockClient"> The Amazon Bedrock client. </param>
private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
{
    Console.WriteLine("List foundation models with no filter.");

    try
    {
        var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
        {
        });

        if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            foreach (var fm in response.ModelSummaries)
            {
                WriteToConsole(fm);
            }
        }
        else
        {
            Console.WriteLine("Something wrong happened");
        }
    }
    catch (AmazonBedrockException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

```
}
```

- Para obtener más información sobre la API, consulta [ListFoundationModels](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon Bedrock Runtime con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con Amazon Bedrock Runtime.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta Llama](#)
- [Mistral AI](#)

Anthropic Claude

Converse

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude con la API de Converse de Bedrock.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Anthropic Claude mediante la API de Converse de Bedrock.

```
// Use the Converse API to send a text message to Anthropic Claude.

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., 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.";

// 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;
}
```

- Para obtener información sobre la API, consulte [Converse](#) en la Referencia de la API de AWS SDK para .NET .

ConverseStream

El siguiente ejemplos de código muestra cómo enviar un mensaje de texto a Anthropic Claude con la API de Converse de Bedrock y procesar el flujo de respuesta en tiempo real.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Anthropic Claude con la API de Converse de Bedrock y procese el flujo de respuesta en tiempo real.

```
// Use the Converse API to send a text message to Anthropic Claude
// 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., 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.";

// 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 obtener más información sobre la API, consulta [ConverseStream](#) la Referencia AWS SDK para .NET de la API.

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude con la API de Invoke Model.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Use la API de Invoke Model para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Anthropic Claude.  
  
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 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["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obtener más información sobre la API, consulta [InvokeModel](#) la Referencia AWS SDK para .NET de la API.

Cohere Command

Converse

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command con la API de Converse de Bedrock.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Cohere Command mediante la API de Converse de Bedrock.

```
// Use the Converse API to send a text message to Cohere Command.

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., 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.";

// 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;
}

```

- Para obtener información sobre la API, consulte [Converse](#) en la Referencia de la API de AWS SDK para .NET .

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command con la API de Converse de Bedrock y procesar el flujo de respuesta en tiempo real.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Cohere Command con la API de Converse de Bedrock y procese el flujo de respuesta en tiempo real.

```
// Use the Converse API to send a text message to Cohere Command
// 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., 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.";

// 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 obtener más información sobre la API, consulta [ConverseStream](#) la Referencia AWS SDK para .NET de la API.

Meta Llama

Converse

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama con la API de Converse de Bedrock.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Meta Llama mediante la API de Converse de Bedrock.

```
// Use the Converse API to send a text message to Meta Llama.

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., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-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;
}

```

- Para obtener información sobre la API, consulte [Converse](#) en la Referencia de la API de AWS SDK para .NET .

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama con la API de Converse de Bedrock y procesar el flujo de respuesta en tiempo real.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Meta Llama con la API de Converse de Bedrock y procese el flujo de respuesta en tiempo real.

```
// Use the Converse API to send a text message to Meta Llama
// 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., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-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 obtener más información sobre la API, consulta [ConverseStream](#) la Referencia AWS SDK para .NET de la API.

Mistral AI

Converse

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral con la API de Converse de Bedrock.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Mistral mediante la API de Converse de Bedrock.

```
// Use the Converse API to send a text message to Mistral.

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., Mistral Large.
var modelId = "mistral.mistral-large-2402-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;  
}
```

- Para obtener información sobre la API, consulte [Converse](#) en la Referencia de la API de AWS SDK para .NET .

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral con la API de Converse de Bedrock y procesar el flujo de respuesta en tiempo real.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Envíe un mensaje de texto a Mistral mediante la API de Converse de Bedrock y procese el flujo de respuesta en tiempo real.

```
// Use the Converse API to send a text message to Mistral
// 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., Mistral Large.
var modelId = "mistral.mistral-large-2402-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 obtener más información sobre la API, consulta [ConverseStream](#) la Referencia AWS SDK para .NET de la API.

CloudFormation ejemplos que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con CloudFormation.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)

Introducción

¿Hola CloudFormation

En el siguiente ejemplo de código se muestra cómo empezar a utilizar CloudFormation.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"In 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());
    if (paginatorForDescribeStacks.Stacks != null)
    {
        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 != null && 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 != null &&
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 using SS0, be sure to install" +
            " the AWSSDK.SSO and AWSSDK.SSO0IDC 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;
}
}
```

- Para obtener más información sobre la API, consulta [DescribeStackResources](#) la Referencia AWS SDK para .NET de la API.

CloudWatch ejemplos que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con CloudWatch.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

¿Hola CloudWatch

En el siguiente ejemplo de código se muestra cómo empezar a utilizar CloudWatch.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.CloudWatch;  
using Amazon.CloudWatch.Model;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;
```

```

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the Amazon CloudWatch 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<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
            host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
            ListMetricsRequest
            {
                Namespace = metricNamespace
            });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
        available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"Metric: {metric.MetricName}");
                Console.WriteLine($"Namespace: {metric.Namespace}");
                Console.WriteLine(
                    $"Dimensions: {string.Join(", ", metric.Dimensions.Select(m =>
                        $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}

```

- Para obtener más información sobre la API, consulta [ListMetrics](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Enumera los espacios de CloudWatch nombres y las métricas.
- Obtener estadísticas para una métrica y para la facturación estimada.
- Crear y actualizar un panel.
- Crear y agregar datos a una métrica.
- Crear y activar una alarma y, a continuación, consultar el historial de alarmas.
- Crear un detector de anomalías.
- Realice una imagen métrica y, luego, limpie los recursos.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
public class CloudWatchScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        To enable billing metrics and statistics for this example, make sure billing
        alerts are enabled for your account:
        https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
        monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics
    */
}
```

This .NET example performs the following tasks:

1. List and select a CloudWatch namespace.
2. List and select a CloudWatch metric.
3. Get statistics for a CloudWatch metric.
4. Get estimated billing statistics for the last week.
5. Create a new CloudWatch dashboard with two metrics.
6. List current CloudWatch dashboards.
7. Create a CloudWatch custom metric and add metric data.
8. Add the custom metric to the dashboard.
9. Create a CloudWatch alarm for the custom metric.
10. Describe current CloudWatch alarms.
11. Get recent data for the custom metric.
12. Add data to the custom metric to trigger the alarm.
13. Wait for an alarm state.
14. Get history for the CloudWatch alarm.
15. Add an anomaly detector.
16. Describe current anomaly detectors.
17. Get and display a metric image.
18. Clean up resources.

*/

```
private static ILogger logger = null!;
private static CloudWatchWrapper _cloudWatchWrapper = null!;
private static IConfiguration _configuration = null!;
private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
private static SingleMetricAnomalyDetector? anomalyDetector = 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<IAmazonCloudWatch>()
                .AddTransient<CloudWatchWrapper>()
        )
        .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<CloudWatchScenario>();

_cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

Console.WriteLine(new string('-', 80));
Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    var selectedNamespace = await SelectNamespace();
    var selectedMetric = await SelectMetric(selectedNamespace);
    await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
    await GetAndDisplayEstimatedBilling();
    await CreateDashboardWithMetrics();
    await ListDashboards();
    await CreateNewCustomMetric();
    await AddMetricToDashboard();
    await CreateMetricAlarm();
    await DescribeAlarms();
    await GetCustomMetricData();
    await AddMetricDataForAlarm();
    await CheckForMetricAlarm();
    await GetAlarmHistory();
    anomalyDetector = await AddAnomalyDetector();
    await DescribeAnomalyDetectors();
    await GetAndOpenMetricImage();
    await CleanupResources();
}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
    await CleanupResources();
}
}

```



```

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"  {i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");
    }

```

```

        var namespaceMetrics = await
            _cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"{i + 1}. {namespaceMetrics[i].MetricName} " +
                $"{string.Join(", :", dimensionsWithValues.Select(d =>
                    d.Value)))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
            namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
        Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"{i + 1}. {_statTypes[i]}");
        }
    }

```

```

    }

    var statisticChoiceNumber = 0;
    while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
    {
        Console.WriteLine(
            "Select a metric statistic by entering a number from the preceding
list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out statisticChoiceNumber);
    }

    var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
    var statisticsList = new List<string> { selectedStatistic };

    var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

    if (!metricStatistics.Any())
    {
        Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
    }

    metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
    for (int i = 0; i < metricStatistics.Count && i < 10; i++)
    {
        var metricStat = metricStatistics[i];
        var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
        Console.WriteLine($"{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get and display estimated billing statistics.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <param name="metric">The CloudWatch metric.</param>

```

```

    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"{t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>

```

```

/// <param name="metric">The CloudWatch metric.</param>
/// <returns>Async task.</returns>
private static async Task CreateDashboardWithMetrics()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
    var dashboardName = _configuration["dashboardName"];
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
    var newDashboardString = JsonSerializer.Serialize(
        newDashboard,
        new JsonSerializerOptions
        {
            DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
        });
    var validationMessages =
        await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

    Console.WriteLine(validationMessages.Any() ? $"\\tValidation messages:" :
null);
    for (int i = 0; i < validationMessages.Count; i++)
    {
        Console.WriteLine($"\\t{i + 1}. {validationMessages[i].Message}");
    }
    Console.WriteLine($"\\tDashboard {dashboardName} was created.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List dashboards.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDashboards()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

    var dashboards = await _cloudWatchWrapper.ListDashboards();

    for (int i = 0; i < dashboards.Count; i++)
    {
        Console.WriteLine($"\\t{i + 1}. {dashboards[i].DashboardName}");
    }
}

```

```

    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create and add data for a new custom metric.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CreateNewCustomMetric()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"7. Create and add data for a new custom metric.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

    var valuesString = string.Join(',', customData.Select(d => d.Value));
    Console.WriteLine($"\\tAdded metric values for for metric {customMetricName}:
\\n\\t{valuesString}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add some metric data using a call to a wrapper class.
/// </summary>
/// <param name="customMetricName">The metric name.</param>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <returns></returns>
private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
    string customMetricNamespace)
{
    List<MetricDatum> customData = new List<MetricDatum>();
    Random rnd = new Random();

    // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
    var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);

```

```

        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
            customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"{Environment.NewLine}Validation messages: " :
            null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"{Environment.NewLine}{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"{Environment.NewLine}Dashboard {dashboardName} updated with metric {customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

```

```

/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
            {
                new() { customMetricNamespace, customMetricName } },
            View = "timeSeries",
            Region = "us-east-1",
            Stat = "Sum",
            Period = 86400,
            YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
            Title = "Custom Metric Widget",
            LiveData = true,
            Sparkline = true,
            Trend = true,
            Stacked = false,
            SetPeriodToTimeRange = false
        }
    });

    var newDashboardString = JsonSerializer.Serialize(newDashboard,
        new JsonSerializerOptions

```



```

        { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
    var validationMessages =
        await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

    return validationMessages;
}

/// <summary>
/// Create a CloudWatch alarm for the new metric.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CreateMetricAlarm()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var alarmName = _configuration["exampleAlarmName"];
    var accountId = _configuration["accountId"];
    var region = _configuration["region"];
    var emailTopic = _configuration["emailTopic"];
    var alarmActions = new List<string>();

    if (GetYesNoResponse(
        $"\\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
    {
        _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
    }

    await _cloudWatchWrapper.PutMetricEmailAlarm(
        "Example metric alarm",
        alarmName,
        ComparisonOperator.GreaterThanOrEqualToThreshold,
        customMetricName,
        customMetricNamespace,
        100,
        alarmActions);
}

```

```

        Console.WriteLine($"\\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {

```

```

        new MetricDataQuery
        {
            AccountId = accountId,
            Id = "m1",
            Label = "Custom Metric Data",
            MetricStat = new MetricStat
            {
                Metric = new Metric
                {
                    MetricName = customMetricName,
                    Namespace = customMetricNamespace,
                },
                Period = 1,
                Stat = "Maximum"
            }
        }
    };

    var metricData = await _cloudWatchWrapper.GetMetricData(
        20,
        true,
        DateTime.UtcNow.AddMinutes(1),
        20,
        query);

    for (int i = 0; i < metricData.Count; i++)
    {
        if (metricData[i].Values != null)
        {
            for (int j = 0; j < metricData[i].Values.Count; j++)
            {
                Console.WriteLine(
                    $"{DateTime.UtcNow.ToString("G")} {metricData[i].Timestamps[j]:G} Value: {metricData[i].Values[j]}");
            }
        }
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add metric data to trigger an alarm.
/// </summary>

```

```

/// <returns>Async task.</returns>
private static async Task AddMetricDataForAlarm()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];
    var nowUtc = DateTime.UtcNow;
    List<MetricDatum> customData = new List<MetricDatum>
    {
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc.AddMinutes(-2)
        },
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc.AddMinutes(-1)
        },
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc
        }
    };
    var valuesString = string.Join(',', customData.Select(d => d.Value));
    Console.WriteLine($"\\tAdded metric values for for metric {customMetricName}:
\\n\\t{valuesString}");
    await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Check for a metric alarm using the DescribeAlarmsForMetric action.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CheckForMetricAlarm()

```

```

{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"13. Checking for an alarm state.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];
    var hasAlarm = false;
    var retries = 10;
    while (!hasAlarm && retries > 0)
    {
        var alarms = await
        _cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
        customMetricName);
        hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
        retries--;
        Thread.Sleep(20000);
    }

    Console.WriteLine(hasAlarm
        ? $"{Environment.NewLine}Alarm state found for {customMetricName}."
        : $"{Environment.NewLine}No Alarm state found for {customMetricName} after 10 retries.");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get history for an alarm.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetAlarmHistory()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"14. Get alarm history.");

    var exampleAlarmName = _configuration["exampleAlarmName"];

    var alarmHistory = await
    _cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

    for (int i = 0; i < alarmHistory.Count; i++)
    {
        var history = alarmHistory[i];
        Console.WriteLine($"{Environment.NewLine}{i + 1}. {history.HistorySummary}, time
        {history.Timestamp:g}");
    }
}

```

```

    }
    if (!alarmHistory.Any())
    {
        Console.WriteLine($"\\tNo alarm history data found for
{exampleAlarmName}.");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add an anomaly detector.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"15. Add an anomaly detector.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var detector = new SingleMetricAnomalyDetector
    {
        MetricName = customMetricName,
        Namespace = customMetricNamespace,
        Stat = "Maximum"
    };
    await _cloudWatchWrapper.PutAnomalyDetector(detector);
    Console.WriteLine($"\\tAdded anomaly detector for metric
{customMetricName}.");

    Console.WriteLine(new string('-', 80));
    return detector;
}

/// <summary>
/// Describe anomaly detectors.
/// </summary>
/// <returns>Async task.</returns>
private static async Task DescribeAnomalyDetectors()
{
    Console.WriteLine(new string('-', 80));

```

```

        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
    }

```

```

        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"Deleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"Cleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
            anomalyDetector != null)
        {
            Console.WriteLine($"Cleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }
    }

```



```

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// 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;
    }
}

```

Métodos envoltorios utilizados por el escenario para CloudWatch las acciones.

```

/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
        ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }
}

```

```

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
    {

```

```

        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
        created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }

    /// <summary>
    /// Get information on a dashboard.
    /// </summary>

```

```

/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}

/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}

/// <summary>
/// Wrapper to add metric data to a CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricData">A data object for the metric data.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricData(string metricNamespace,
    List<MetricDatum> metricData)
{
    var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
        new PutMetricDataRequest()
        {

```

```

        MetricData = metricData,
        Namespace = metricNamespace,
    });

    return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get an image for a metric graphed over time.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metric">The name of the metric.</param>
/// <param name="stat">The name of the stat to chart.</param>
/// <param name="period">The period to use for the chart.</param>
/// <returns>A memory stream for the chart image.</returns>
public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
{
    var metricImageWidget = new
    {
        title = "Example Metric Graph",
        view = "timeSeries",
        stacked = false,
        period = period,
        width = 1400,
        height = 600,
        metrics = new List<List<object>>
        {
            new() { metricNamespace, metric, new { stat } } }
    };

    var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
    var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
        new GetMetricWidgetImageRequest()
        {
            MetricWidget = metricImageWidgetString
        });

    return imageResponse.MetricWidgetImage;
}

/// <summary>
/// Save a metric image to a file.
/// </summary>
/// <param name="memoryStream">The MemoryStream for the metric image.</param>

```

```

    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {

```

```

        StartTimeUtc = startTimeUtc,
        EndTimeUtc = endDateUtc.Value,
        LabelOptions = new LabelOptions { Timezone = timeZoneString },
        ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
        MaxDatapoints = maxDataPoints,
        MetricDataQueries = dataQueries,
    });

    if (paginatedMetricData.MetricDataResults != null)
    {
        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
    }

    return metricData;
}

/// <summary>
/// Add a metric alarm to send an email when the metric passes a threshold.
/// </summary>
/// <param name="alarmDescription">A description of the alarm.</param>
/// <param name="alarmName">The name for the alarm.</param>
/// <param name="comparison">The type of comparison to use.</param>
/// <param name="metricName">The name of the metric for the alarm.</param>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="threshold">The threshold value for the alarm.</param>
/// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
    string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
{
    try
    {
        var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
            new PutMetricAlarmRequest()
            {
                AlarmActions = alarmActions,
                AlarmDescription = alarmDescription,

```

```

        AlarmName = alarmName,
        ComparisonOperator = comparison,
        Threshold = threshold,
        Namespace = metricNamespace,
        MetricName = metricName,
        EvaluationPeriods = 1,
        Period = 10,
        Statistic = new Statistic("Maximum"),
        DatapointsToAlarm = 1,
        TreatMissingData = "ignore"
    });
    return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
}
catch (LimitExceededException lex)
{
    _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
}

return false;
}

/// <summary>
/// Add specific email actions to a list of action strings for a CloudWatch
alarm.
/// </summary>
/// <param name="accountId">The AccountId for the alarm.</param>
/// <param name="region">The region for the alarm.</param>
/// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
/// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
/// <returns>A list of string actions for an alarm.</returns>
public List<string> AddEmailAlarmAction(string accountId, string region,
    string emailTopicName, List<string>? alarmActions = null)
{
    alarmActions ??= new List<string>();
    var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
    alarmActions.Add(snsAlarmAction);
    return alarmActions;
}

/// <summary>
/// Describe the current alarms, optionally filtered by state.

```



```

    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>

```

```

    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
        _amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)

```

```

{
    var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
    new DisableAlarmActionsRequest()
    {
        AlarmNames = alarmNames
    });

    return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Enable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableAlarmActions(List<string> alarmNames)
{
    var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
    new EnableAlarmActionsRequest()
    {
        AlarmNames = alarmNames
    });

    return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Add an anomaly detector for a single metric.
/// </summary>
/// <param name="anomalyDetector">A single metric anomaly detector.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
    new PutAnomalyDetectorRequest()
    {
        SingleMetricAnomalyDetector = anomalyDetector
    });

    return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
}

```

```

    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
        _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
        _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });
    }

```

```

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
        _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}

```

Ejemplo de valores de settings.json para este escenario:

```

{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by\nexample code.\n"
        }
      }
    ]
  }
}

```

```

    }
  },
  {
    "height": 8,
    "width": 8,
    "y": 0,
    "x": 6,
    "type": "metric",
    "properties": {
      "metrics": [
        [
          "AWS/Billing",
          "EstimatedCharges",
          "Currency",
          "USD",
          { "region": "us-east-1" }
        ]
      ],
      "view": "timeSeries",
      "region": "us-east-1",
      "stat": "Maximum",
      "period": 86400,
      "yAxis": {
        "left": {
          "min": 0,
          "max": 100
        }
      },
    },
    "stacked": false,
    "title": "Estimated Billing",
    "setPeriodToTimeRange": false,
    "liveData": true,
    "sparkline": true,
    "trend": true
  }
},
{
  "height": 8,
  "width": 8,
  "y": 0,
  "x": 14,
  "type": "metric",
  "properties": {
    "metrics": [

```

```

        [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
"Service", "CloudWatch", "Class", "None" ],
        [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
        [ "...", "GetMetricData", ".", ".", ".", "." ],
        [ "...", "PutDashboard", ".", ".", ".", "." ],
        [ "...", "PutMetricData", ".", ".", ".", "." ]
    ],
    "view": "timeSeries",
    "yAxis": {
        "left": {
            "min": 0,
            "max": 200
        }
    },
    "stacked": false,
    "region": "us-east-1",
    "stat": "Sum",
    "period": 300,
    "title": "CloudWatch Usage",
    "setPeriodToTimeRange": false,
    "liveData": true,
    "sparkline": true,
    "trend": true
    }
}
]
}
}

```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [DeleteAlarms](#)
 - [DeleteAnomalyDetector](#)
 - [DeleteDashboards](#)
 - [DescribeAlarmHistory](#)
 - [DescribeAlarms](#)
 - [DescribeAlarmsForMetric](#)
 - [DescribeAnomalyDetectors](#)

- [GetMetricData](#)
- [GetMetricStatistics](#)
- [GetMetricWidgetImage](#)
- [ListMetrics](#)
- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)
- [PutMetricData](#)

Acciones

DeleteAlarms

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteAlarms.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```


- Para obtener más información sobre la API, consulta [DeleteAlarms](#) la Referencia AWS SDK para .NET de la API.

DeleteAnomalyDetector

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteAnomalyDetector.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete a single metric anomaly detector.
/// </summary>
/// <param name="anomalyDetector">The anomaly detector to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var deleteAnomalyDetectorResponse = await
        _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

    return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [DeleteAnomalyDetector](#) la Referencia AWS SDK para .NET de la API.

DeleteDashboards

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteDashboards.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete a list of CloudWatch dashboards.
/// </summary>
/// <param name="dashboardNames">List of dashboard names to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteDashboards(List<string> dashboardNames)
{
    var deleteDashboardsResponse = await
        _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

    return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [DeleteDashboards](#) la Referencia AWS SDK para .NET de la API.

DescribeAlarmHistory

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeAlarmHistory.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Describe the history of an alarm for a number of days in the past.
/// </summary>
/// <param name="alarmName">The name of the alarm.</param>
/// <param name="historyDays">The number of days in the past.</param>
/// <returns>The list of alarm history data.</returns>
public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
{
    List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
    var paginatedAlarmHistory =
        _amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

    await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
    {
        alarmHistory.Add(data);
    }
    return alarmHistory;
}
```

- Para obtener más información sobre la API, consulta [DescribeAlarmHistory](#) la Referencia AWS SDK para .NET de la API.

DescribeAlarms

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeAlarms`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Describe the current alarms, optionally filtered by state.
/// </summary>
/// <param name="stateValue">Optional filter for alarm state.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
{
    List<MetricAlarm> alarms = new List<MetricAlarm>();
    var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
        new DescribeAlarmsRequest()
        {
            StateValue = stateValue
        });

    await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
    {
        alarms.Add(data);
    }
    return alarms;
}
```

- Para obtener más información sobre la API, consulta [DescribeAlarms](#) la Referencia AWS SDK para .NET de la API.

DescribeAlarmsForMetric

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeAlarmsForMetric`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Describe the current alarms for a specific metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The name of the metric.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
{
    var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
        new DescribeAlarmsForMetricRequest()
        {
            Namespace = metricNamespace,
            MetricName = metricName
        });

    return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
}
```

- Para obtener más información sobre la API, consulta [DescribeAlarmsForMetric](#) la Referencia AWS SDK para .NET de la API.

DescribeAnomalyDetectors

En el siguiente ejemplo de código, se muestra cómo utilizar `DescribeAnomalyDetectors`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Describe anomaly detectors for a metric and namespace.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The metric of the anomaly detectors.</param>
/// <returns>The list of detectors.</returns>
public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
{
    List<AnomalyDetector> detectors = new List<AnomalyDetector>();
    var paginatedDescribeAnomalyDetectors =
    _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
        new DescribeAnomalyDetectorsRequest()
        {
            MetricName = metricName,
            Namespace = metricNamespace
        });

    await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
    {
        detectors.Add(data);
    }

    return detectors;
}
```

- Para obtener más información sobre la API, consulta [DescribeAnomalyDetectors](#) la Referencia AWS SDK para .NET de la API.

DisableAlarmActions

En el siguiente ejemplo de código, se muestra cómo utilizar `DisableAlarmActions`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Disable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableAlarmActions(List<string> alarmNames)
{
    var disableAlarmActionsResult = await
        _amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

    return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [DisableAlarmActions](#) la Referencia AWS SDK para .NET de la API.

EnableAlarmActions

En el siguiente ejemplo de código, se muestra cómo utilizar `EnableAlarmActions`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Enable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableAlarmActions(List<string> alarmNames)
{
    var enableAlarmActionsResult = await
        _amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

    return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [EnableAlarmActions](#) la Referencia AWS SDK para .NET de la API.

GetDashboard

En el siguiente ejemplo de código, se muestra cómo utilizar GetDashboard.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).


```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- Para obtener más información sobre la API, consulta [GetDashboard](#) la Referencia AWS SDK para .NET de la API.

GetMetricData

En el siguiente ejemplo de código, se muestra cómo utilizar GetMetricData.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get data for CloudWatch metrics.
/// </summary>
/// <param name="minutesOfData">The number of minutes of data to include.</param>
/// <param name="useDescendingTime">True to return the data descending by time.</param>
/// <param name="endDateUtc">The end date for the data, in UTC.</param>
```

```

    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

```

- Para obtener más información sobre la API, consulta [GetMetricData](#) la Referencia AWS SDK para .NET de la API.

GetMetricStatistics

En el siguiente ejemplo de código, se muestra cómo utilizar `GetMetricStatistics`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Get billing statistics using a call to a wrapper class.
/// </summary>
/// <returns>A collection of billing statistics.</returns>
private static async Task<List<Datapoint>> SetupBillingStatistics()
{
    // Make a request for EstimatedCharges with a period of one day for the past
    seven days.
    var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
        "AWS/Billing",
        "EstimatedCharges",
        new List<string>() { "Maximum" },
        new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
        7,
        86400);

    billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

    return billingStatistics;
}

/// <summary>
/// Wrapper to get statistics for a specific CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The name of the metric.</param>

```

```

/// <param name="statistics">The list of statistics to include.</param>
/// <param name="dimensions">The list of dimensions to include.</param>
/// <param name="days">The number of days in the past to include.</param>
/// <param name="period">The period for the data.</param>
/// <returns>A list of DataPoint objects for the statistics.</returns>
public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
    string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
{
    var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
        new GetMetricStatisticsRequest()
        {
            Namespace = metricNamespace,
            MetricName = metricName,
            Dimensions = dimensions,
            Statistics = statistics,
            StartTimeUtc = DateTime.UtcNow.AddDays(-days),
            EndTimeUtc = DateTime.UtcNow,
            Period = period
        });

    return metricStatistics.Datapoints ?? new List<Datapoint>();
}

```

- Para obtener más información sobre la API, consulta [GetMetricStatistics](#) la Referencia AWS SDK para .NET de la API.

GetMetricWidgetImage

En el siguiente ejemplo de código, se muestra cómo utilizar `GetMetricWidgetImage`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
```

```

    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
    }

```

```
var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
    metricFileName);
return filePath;
}
```

- Para obtener más información sobre la API, consulta [GetMetricWidgetImage](#) la Referencia AWS SDK para .NET de la API.

ListDashboards

En el siguiente ejemplo de código, se muestra cómo utilizar ListDashboards.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}
```

- Para obtener más información sobre la API, consulta [ListDashboards](#) la Referencia AWS SDK para .NET de la API.

ListMetrics

En el siguiente ejemplo de código, se muestra cómo utilizar ListMetrics.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// List metrics available, optionally within a namespace.
/// </summary>
/// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
/// <param name="filter">Optional dimension filter.</param>
/// <param name="metricName">Optional metric name filter.</param>
/// <returns>The list of metrics.</returns>
public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
{
    var results = new List<Metric>();
    var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
        new ListMetricsRequest
        {
            Namespace = metricNamespace,
            Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
            MetricName = metricName
        });
    // Get the entire list using the paginator.
    await foreach (var metric in paginateMetrics.Metrics)
    {
        results.Add(metric);
    }

    return results;
}

```

```
}
```

- Para obtener más información sobre la API, consulta [ListMetrics](#) la Referencia AWS SDK para .NET de la API.

PutAnomalyDetector

En el siguiente ejemplo de código, se muestra cómo utilizar PutAnomalyDetector.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Add an anomaly detector for a single metric.
/// </summary>
/// <param name="anomalyDetector">A single metric anomaly detector.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var putAlarmDetectorResult = await
    _amazonCloudWatch.PutAnomalyDetectorAsync(
        new PutAnomalyDetectorRequest()
        {
            SingleMetricAnomalyDetector = anomalyDetector
        });

    return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta [PutAnomalyDetector](#) la Referencia AWS SDK para .NET de la API.

PutDashboard

En el siguiente ejemplo de código, se muestra cómo utilizar PutDashboard.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
            {
                new() { customMetricNamespace, customMetricName } },
            View = "timeSeries",
            Region = "us-east-1",
            Stat = "Sum",
            Period = 86400,
        }
    });
}

```

```

        YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
        Title = "Custom Metric Widget",
        LiveData = true,
        Sparkline = true,
        Trend = true,
        Stacked = false,
        SetPeriodToTimeRange = false
    }
});

var newDashboardString = JsonSerializer.Serialize(newDashboard,
    new JsonSerializerOptions
    { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
var validationMessages =
    await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

    return validationMessages;
}

/// <summary>
/// Wrapper to create or add to a dashboard with metrics.
/// </summary>
/// <param name="dashboardName">The name for the dashboard.</param>
/// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
/// <returns>A list of validation messages for the dashboard.</returns>
public async Task<List<DashboardValidationMessage>> PutDashboard(string
dashboardName,
    string dashboardBody)
{
    // Updating a dashboard replaces all contents.
    // Best practice is to include a text widget indicating this dashboard was
    created programmatically.
    var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
        new PutDashboardRequest()
        {
            DashboardName = dashboardName,
            DashboardBody = dashboardBody
        });

    return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
}

```

- Para obtener más información sobre la API, consulta [PutDashboard](#) la Referencia AWS SDK para .NET de la API.

PutMetricAlarm

En el siguiente ejemplo de código, se muestra cómo utilizar PutMetricAlarm.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Add a metric alarm to send an email when the metric passes a threshold.
/// </summary>
/// <param name="alarmDescription">A description of the alarm.</param>
/// <param name="alarmName">The name for the alarm.</param>
/// <param name="comparison">The type of comparison to use.</param>
/// <param name="metricName">The name of the metric for the alarm.</param>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="threshold">The threshold value for the alarm.</param>
/// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
    string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
{
    try
    {
        var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
            new PutMetricAlarmRequest()
            {
                AlarmActions = alarmActions,

```

```

        AlarmDescription = alarmDescription,
        AlarmName = alarmName,
        ComparisonOperator = comparison,
        Threshold = threshold,
        Namespace = metricNamespace,
        MetricName = metricName,
        EvaluationPeriods = 1,
        Period = 10,
        Statistic = new Statistic("Maximum"),
        DatapointsToAlarm = 1,
        TreatMissingData = "ignore"
    });
    return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
}
catch (LimitExceededException lex)
{
    _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
}

return false;
}

/// <summary>
/// Add specific email actions to a list of action strings for a CloudWatch
alarm.
/// </summary>
/// <param name="accountId">The AccountId for the alarm.</param>
/// <param name="region">The region for the alarm.</param>
/// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
/// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
/// <returns>A list of string actions for an alarm.</returns>
public List<string> AddEmailAlarmAction(string accountId, string region,
    string emailTopicName, List<string>? alarmActions = null)
{
    alarmActions ??= new List<string>();
    var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
    alarmActions.Add(snsAlarmAction);
    return alarmActions;
}

```

- Para obtener más información sobre la API, consulta [PutMetricAlarm](#) la Referencia AWS SDK para .NET de la API.

PutMetricData

En el siguiente ejemplo de código, se muestra cómo utilizar PutMetricData.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Add some metric data using a call to a wrapper class.
/// </summary>
/// <param name="customMetricName">The metric name.</param>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <returns></returns>
private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
    string customMetricNamespace)
{
    List<MetricDatum> customData = new List<MetricDatum>();
    Random rnd = new Random();

    // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
    var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
    for (int i = 0; i < 10; i++)
    {
        var metricValue = rnd.Next(0, 100);
        customData.Add(
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = metricValue,
                TimestampUtc = utcNowMinus15.AddMinutes(i)
            }
        )
    }
}
```

```

        });
    }

    await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
    return customData;
}

/// <summary>
/// Wrapper to add metric data to a CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricData">A data object for the metric data.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricData(string metricNamespace,
    List<MetricDatum> metricData)
{
    var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
        new PutMetricDataRequest()
        {
            MetricData = metricData,
            Namespace = metricNamespace,
        });

    return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
}

```

- Para obtener más información sobre la API, consulta [PutMetricData](#) la Referencia AWS SDK para .NET de la API.

CloudWatch Registra ejemplos con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con CloudWatch registros.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Acciones](#)
- [Escenarios](#)

Acciones

GetQueryResults

En el siguiente ejemplo de código, se muestra cómo utilizar `GetQueryResults`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Gets the results of a CloudWatch Logs Insights query.
/// </summary>
/// <param name="queryId">The ID of the query.</param>
/// <returns>The query results response.</returns>
public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
{
    try
    {
        var request = new GetQueryResultsRequest
        {
            QueryId = queryId
        };

        var response = await
            _amazonCloudWatchLogs.GetQueryResultsAsync(request);
        return response;
    }
    catch (ResourceNotFoundException ex)
```

```

    {
        _logger.LogError($"Query not found: {ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while getting query results:
{ex.Message}");
        return null;
    }
}

```

- Para obtener más información sobre la API, consulta [GetQueryResults](#) la Referencia AWS SDK para .NET de la API.

StartQuery

En el siguiente ejemplo de código, se muestra cómo utilizar StartQuery.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Starts a CloudWatch Logs Insights query.
/// </summary>
/// <param name="logGroupName">The name of the log group to query.</param>
/// <param name="queryString">The CloudWatch Logs Insights query string.</param>
/// <param name="startTime">The start time for the query (seconds since
epoch).</param>
/// <param name="endTime">The end time for the query (seconds since epoch).</
param>
/// <param name="limit">The maximum number of results to return.</param>
/// <returns>The query ID if successful, null otherwise.</returns>
public async Task<string?> StartQueryAsync(
    string logGroupName,

```



```
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query: {ex.Message}");
            return null;
        }
    }
}
```

- Para obtener más información sobre la API, consulta [StartQuery](#) la Referencia AWS SDK para .NET de la API.

Escenarios

Ejecución de una consulta de gran tamaño

El siguiente ejemplo de código muestra cómo usar CloudWatch los registros para consultar más de 10 000 registros.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Este es el flujo de trabajo principal que muestra el escenario de consultas de gran tamaño.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET code example performs the following tasks for the CloudWatch Logs
    Large Query workflow:

    1. Prepare the Application:
        - Prompt the user to deploy CloudFormation stack and generate sample logs.
        - Deploy the CloudFormation template for resource creation.
        - Generate 50,000 sample log entries using CloudWatch Logs API.
```

- Wait 5 minutes for logs to be fully ingested.

2. Execute Large Query:

- Perform recursive queries to retrieve all logs using binary search.
- Display progress for each query executed.
- Show total execution time and logs found.

3. Clean up:

- Prompt the user to delete the CloudFormation stack and all resources.
- Destroy the CloudFormation stack and wait until removed.

*/

```
public static ILogger<LargeQueryWorkflow> _logger = null!;
public static CloudWatchLogsWrapper _wrapper = null!;
public static IAmazonCloudFormation _amazonCloudFormation = null!;

private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
private static string _logStreamName = "stream1";
private static long _queryStartDate;
private static long _queryEndDate;

public static bool _interactive = true;
public static string _stackName = "CloudWatchLargeQueryStack";
private static string _stackResourcePath = "../..../..../..../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

public static async Task Main(string[] args)
{
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter("Microsoft", LogLevel.Information))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatchLogs>()
                .AddAWSService<IAmazonCloudFormation>()
                .AddTransient<CloudWatchLogsWrapper>()
            )
        .Build();

    if (_interactive)
    {
        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LargeQueryWorkflow>();
    }
}
```

```

        _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
        _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
    }

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
    Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
    Console.WriteLine("the 10,000 result limit.");
    Console.WriteLine();

    try
    {
        Console.WriteLine(new string('-', 80));
        var prepareSuccess = await PrepareApplication();
        Console.WriteLine(new string('-', 80));

        if (prepareSuccess)
        {
            Console.WriteLine(new string('-', 80));
            await ExecuteLargeQuery();
            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("CloudWatch Logs Large Query scenario completed.");
}

/// <summary>

```

```

/// Runs the scenario workflow. Used for testing.
/// </summary>
public static async Task RunScenario()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
    Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
    Console.WriteLine("the 10,000 result limit.");
    Console.WriteLine();

    try
    {
        Console.WriteLine(new string('-', 80));
        var prepareSuccess = await PrepareApplication();
        Console.WriteLine(new string('-', 80));

        if (prepareSuccess)
        {
            Console.WriteLine(new string('-', 80));
            await ExecuteLargeQuery();
            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("CloudWatch Logs Large Query 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...");
    Console.WriteLine();

    try
    {
        var deployStack = !_interactive || GetYesNoResponse(
            "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

        if (deployStack)
        {
            if (!_interactive)
            {
                Console.Write(
                    $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                string? inputPath = Console.ReadLine();
                if (!string.IsNullOrEmpty(inputPath))
                {
                    _stackResourcePath = inputPath;
                }
            }

            _stackName = PromptUserForStackName();

            var deploySuccess = await DeployCloudFormationStack(_stackName);

            if (deploySuccess)
            {
                Console.WriteLine();
                Console.WriteLine("Generating 50,000 sample log entries...");
                var generateSuccess = await GenerateSampleLogs();

                if (generateSuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                    await WaitWithCountdown(300);
                }
            }
        }
    }
}

```

```

        Console.WriteLine("Application preparation complete.");
        return true;
    }
}
else
{
    _logGroupName = PromptUserForInput("Enter the log group name ",
    _logGroupName);
    _logStreamName = PromptUserForInput("Enter the log stream name ",
    _logStreamName);

    var startDateMs = PromptUserForLong("Enter the query start date
(millisecons since epoch): ");
    var endDateMs = PromptUserForLong("Enter the query end date
(millisecons since epoch): ");

    _queryStartDate = startDateMs / 1000;
    _queryEndDate = endDateMs / 1000;

    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>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName)
{
    Console.WriteLine($"\\nDeploying CloudFormation stack: {stackName}");

    try

```

```

    {
        var request = new CreateStackRequest
        {
            StackName = stackName,
            TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
        };

        var response = await _amazonCloudFormation.CreateStackAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

            bool stackCreated = await WaitForStackCompletion(response.StackId);

            if (stackCreated)
            {
                Console.WriteLine("CloudFormation stack created successfully.");
                return true;
            }
            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);
    }
    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="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 = 30;
    const int retryDelay = 10000;

    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)
            {
                return true;
            }
            if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
            {
                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>
/// Generates sample logs directly using CloudWatch Logs API.
/// Creates 50,000 log entries spanning 5 minutes.
/// </summary>
/// <returns>True if logs were generated successfully.</returns>
private static async Task<bool> GenerateSampleLogs()
{
    const int totalEntries = 50000;
    const int entriesPerBatch = 10000;
    const int fiveMinutesMs = 5 * 60 * 1000;

    try
    {
        // Calculate timestamps
        var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
        var timestampIncrement = fiveMinutesMs / totalEntries;

        Console.WriteLine($"Generating {totalEntries} log entries...");

        var entryCount = 0;
        var currentTimestamp = startTimeMs;
        var numBatches = totalEntries / entriesPerBatch;

        // Generate and upload logs in batches
        for (int batchNum = 0; batchNum < numBatches; batchNum++)
        {
            var logEvents = new List<InputLogEvent>();

            for (int i = 0; i < entriesPerBatch; i++)
            {
                logEvents.Add(new InputLogEvent
                {

```

```

        Timestamp =
DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
        Message = $"Entry {entryCount}"
    });

    entryCount++;
    currentTimestamp += timestampIncrement;
}

// Upload batch
var success = await _wrapper.PutLogEventsAsync(_logGroupName,
_logStreamName, logEvents);
if (!success)
{
    _logger.LogError($"Failed to upload batch {batchNum + 1}/{
numBatches}");
    return false;
}

Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
}

// Set query date range (convert milliseconds to seconds for query API)
_queryStartDate = startTimeMs / 1000;
_queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

Console.WriteLine($"Query start date:
{DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
Console.WriteLine($"Query end date:
{DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

return true;
}
catch (Exception ex)
{
    _logger.LogError(ex, "An error occurred while generating sample logs.");
    return false;
}
}

/// <summary>
/// Executes the large query workflow.
/// </summary>

```

```

public static async Task ExecuteLargeQuery()
{
    Console.WriteLine("Starting recursive query to retrieve all logs...");
    Console.WriteLine();

    var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
    if (queryLimit > 10000) queryLimit = 10000;

    var queryString = "fields @timestamp, @message | sort @timestamp asc";

    var stopwatch = Stopwatch.StartNew();
    var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
    stopwatch.Stop();

    Console.WriteLine();
    Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
    Console.WriteLine($"Total logs found: {allResults.Count}");

    // Check for duplicates
    Console.WriteLine();
    Console.WriteLine("Checking for duplicate logs...");
    var duplicates = FindDuplicateLogs(allResults);
    if (duplicates.Count > 0)
    {
        Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
        Console.WriteLine("Duplicate entries (showing first 10):");
        foreach (var dup in duplicates.Take(10))
        {
            Console.WriteLine($" [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
        }

        var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
        Console.WriteLine($"Unique logs: {uniqueCount}");
    }
    else
    {
        Console.WriteLine("No duplicates found. All logs are unique.");
    }
    Console.WriteLine();
}

```

```

        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

```

```

        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($" -> Returning {results.Count} logs (less than limit of {limit})");
            return results;
        }

        Console.WriteLine($" -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($" -> No timestamp found in results. Returning {results.Count} logs.");
            return results;
        }

        Console.WriteLine($" -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($" -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($" -> Offset timestamp (last + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ} ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping with logs in the same second
        // This ensures we don't re-query logs that share the same second as the last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second

```

```

        Console.WriteLine($" -> Adjusted to next full second:
{offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-MM-ddTHH:mm:ss.fffZ})");
    }

    Console.WriteLine($" -> Comparing: offsetLastLogTime={offsetLastLogTime}s
vs endTime={endTime}s");
    Console.WriteLine($" -> End time as date:
{DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

    // Check if there's any time range left to query
    if (offsetLastLogTime >= endTime)
    {
        Console.WriteLine($" -> No time range left to query. Offset time
({offsetLastLogTime}s) >= end time ({endTime}s");
        return results;
    }

    // Split the remaining date range in half
    var (range1Start, range1End, range2Start, range2End) =
SplitDateRange(offsetLastLogTime, endTime);

    var range1StartDate =
DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
    var range1EndDate =
DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
    var range2StartDate =
DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
    var range2EndDate =
DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

    Console.WriteLine($" -> Splitting remaining range:");
    Console.WriteLine($"    Range 1: {range1StartDate} ({range1Start}s) to
{range1EndDate} ({range1End}s");
    Console.WriteLine($"    Range 2: {range2StartDate} ({range2Start}s) to
{range2EndDate} ({range2End}s");

    // Query both halves recursively
    Console.WriteLine($" -> Querying range 1...");
    var results1 = await PerformLargeQuery(logGroupName, queryString,
range1Start, range1End, limit);
    Console.WriteLine($" -> Range 1 returned {results1.Count} logs");

```

```

        Console.WriteLine($" -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($" -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($" -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {

```



```

        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }

```

```

    /// <summary>
    /// Cleans up the resources created during the scenario.
    /// </summary>
    public static async Task<bool> Cleanup()
    {
        var cleanup = !_interactive || GetYesNoResponse(
            "Do you want to delete the CloudFormation stack and all resources? (y/n)");

        if (cleanup)
        {
            try
            {
                var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
                return stackDeleteSuccess;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
                return false;
            }
        }

        Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
        _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
        return true;
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    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>
/// Waits for the stack to be deleted.
/// </summary>
private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
{
    int retryCount = 0;
    const int maxRetries = 30;
    const int retryDelay = 10000;

    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)
        {
            return await DeleteCloudFormationStack(stackName, true);
        }
    }
    catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
        "ValidationError")
    {
        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>
/// Waits with a countdown display.
/// </summary>
private static async Task WaitWithCountdown(int seconds)
{
    for (int i = seconds; i > 0; i--)
    {
        Console.Write($"\\rWaiting: {i} seconds remaining... ");
        await Task.Delay(1000);
    }
    Console.WriteLine($"\\rWait complete.                ");
}

```

```

    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    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>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrEmpty(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {

```

```

        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrEmpty(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrEmpty(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }
}

```

```

    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}

```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [GetQueryResults](#)
 - [StartQuery](#)

Ejemplos de proveedores de identidad de Amazon Cognito con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon Cognito Identity Provider.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Acciones](#)

Acciones

ListUserPools

En el siguiente ejemplo de código, se muestra cómo utilizar ListUserPools.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <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;
}
```

- Para obtener más información sobre la API, consulta [ListUserPools](#) la Referencia AWS SDK para .NET de la API.

AWS Control Tower ejemplos que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con AWS Control Tower.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

¿Hola AWS Control Tower

En el siguiente ejemplo de código se muestra cómo empezar a utilizar AWS Control Tower.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        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<IAmazonControlTower>()
        )
        .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<HelloControlTower>();

var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

Console.Clear();
Console.WriteLine("Hello, AWS Control Tower! Let's list available
baselines:");
Console.WriteLine();

var baselines = new List<BaselineSummary>();

try
{
    var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
ListBaselinesRequest());

    await foreach (var response in baselinesPaginator.Responses)
    {
        baselines.AddRange(response.Baselines);
    }

    Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
    foreach (var baseline in baselines)
    {
        Console.WriteLine($"{baseline.Name}");
    }
}
catch (Amazon.ControlTower.Model.AccessDeniedException)
{
    Console.WriteLine("Access denied. Please ensure you have the necessary
permissions.");
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred: {ex.Message}");
}
}

```

```
}
```

- Para obtener más información sobre la API, consulta [ListBaselines](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Mostrar las zonas de aterrizaje.
- Mostrar, habilitar, obtener, restablecer y deshabilitar líneas de base.
- Mostrar, habilitar, obtener y deshabilitar controles.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute un escenario interactivo que demuestre AWS Control Tower las funciones.

```
using Amazon.ControlCatalog;  
using Amazon.ControlTower;  
using Amazon.ControlTower.Model;  
using Amazon.Organizations;  
using Amazon.Organizations.Model;  
using Amazon.SecurityToken;  
using Amazon.SecurityToken.Model;  
using ControlTowerActions;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;  
using Microsoft.Extensions.Logging;
```

```

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
            {
                services.AddAWSService<IAmazonControlTower>()
                    .AddAWSService<IAmazonControlCatalog>()
                    .AddAWSService<IAmazonOrganizations>()
                    .AddAWSService<IAmazonSecurityTokenService>()
                    .AddTransient<ControlTowerWrapper>()
            })
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>

```

```

public static async Task RunScenario()
{
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
    Console.WriteLine(new string('-', 88));
    Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
    Console.WriteLine("managing baselines, and working with controls.");

    try
    {
        var accountId = (await stsClient!.GetCallerIdentityAsync(new
GetCallerIdentityRequest())).Account;
        Console.WriteLine($"Account ID: {accountId}");

        Console.WriteLine("\nSome demo operations require the use of a landing
zone.");
        Console.WriteLine("You can use an existing landing zone or opt out of
these operations in the demo.");
        Console.WriteLine("For instructions on how to set up a landing zone,");
        Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

        // List available landing zones
        var landingZones = await wrapper!.ListLandingZonesAsync();
        if (landingZones.Count > 0)
        {
            Console.WriteLine("\nAvailable Landing Zones:");
            for (int i = 0; i < landingZones.Count; i++)
            {
                Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
            }

            Console.WriteLine($"\nDo you want to use the first landing zone in the
list ({landingZones[0].Arn})? (y/n): ");
            if (GetUserConfirmation())
            {
                useLandingZone = true;
                Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                ouArn = await SetupOrganizationAsync();
            }
        }
    }
}

```

```

// Managing Baselines
Console.WriteLine("\nManaging Baselines:");
var baselines = await wrapper.ListBaselinesAsync();
Console.WriteLine("\nListing available Baselines:");
BaselineSummary? controlTowerBaseline = null;
foreach (var baseline in baselines)
{
    if (baseline.Name == "AWSControlTowerBaseline")
        controlTowerBaseline = baseline;
    Console.WriteLine($" - {baseline.Name}");
}

EnabledBaselineSummary? identityCenterBaseline = null;
string? baselineArn = null;

if (useLandingZone && ouArn != null)
{
    Console.WriteLine("\nListing enabled baselines:");
    var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
    foreach (var baseline in enabledBaselines)
    {
        if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
            identityCenterBaseline = baseline;
        Console.WriteLine($" - {baseline.BaselineIdentifier}");
    }

    if (controlTowerBaseline != null)
    {
        Console.WriteLine("\nDo you want to enable the Control Tower
Baseline? (y/n): ");
        if (GetUserConfirmation())
        {
            Console.WriteLine("\nEnabling Control Tower Baseline.");
            var icBaselineArn = identityCenterBaseline?.Arn;
            baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
            var alreadyEnabled = false;
            if (baselineArn != null)
            {
                Console.WriteLine($"Enabled baseline ARN:
{baselineArn}");
            }
            else

```

```

        {
            // Find the enabled baseline
            foreach (var enabled in enabledBaselines)
            {
                if (enabled.BaselineIdentifier ==
controlTowerBaseline.Arn)
                {
                    baselineArn = enabled.Arn;
                    alreadyEnabled = true;
                    Console.WriteLine("No change, the selected
baseline was already enabled.");
                    break;
                }
            }

            if (baselineArn != null)
            {
                Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                if (GetUserConfirmation())
                {
                    Console.WriteLine($"Resetting Control Tower
Baseline: {baselineArn}");
                    var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                    Console.WriteLine($"Reset baseline operation id:
{operationId}");
                }

                Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                if (GetUserConfirmation())
                {
                    Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                    var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                    Console.WriteLine($"Disabled baseline operation id:
{operationId}");

                    if (alreadyEnabled)
                    {
                        Console.WriteLine($"Re-enabling Control Tower
Baseline: {baselineArn}");

```



```

// Re-enable the Control Tower baseline if it
was originally enabled.
        await wrapper.EnableBaselineAsync(ouArn,
            controlTowerBaseline.Arn, "5.0",
icBaselineArn ?? "");
    }
    }
    }
}

// Managing Controls
Console.WriteLine("\nManaging Controls:");
var controls = await wrapper.ListControlsAsync();
Console.WriteLine("\nListing first 5 available Controls:");
for (int i = 0; i < Math.Min(5, controls.Count); i++)
{
    Console.WriteLine($"{i + 1}. {controls[i].Name} -
{controls[i].Arn}");
}

if (useLandingZone && ouArn != null)
{
    var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
    Console.WriteLine("\nListing enabled controls:");
    for (int i = 0; i < enabledControls.Count; i++)
    {
        Console.WriteLine($"{i + 1}.
{enabledControls[i].ControlIdentifier}");
    }

    // Find first non-enabled control
    var enabledControlArns = enabledControls.Select(c =>
c.Arn).ToHashSet();
    var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn)).Arn;

    if (controlArn != null)
    {
        Console.WriteLine($"Do you want to enable the control
{controlArn}? (y/n): ");
        if (GetUserConfirmation())
        {

```

```

        Console.WriteLine($"\\nEnabling control: {controlArn}");
        var operationId = await
wrapper.EnableControlAsync(controlArn, ouArn);
        if (operationId != null)
        {
            Console.WriteLine($"Enabled control with operation id:
{operationId}");

            Console.Write("\\nDo you want to disable the control? (y/
n): ");

            if (GetUserConfirmation())
            {
                Console.WriteLine("\\nDisabling the control...");
                var disableOpId = await
wrapper.DisableControlAsync(controlArn, ouArn);
                Console.WriteLine($"Disable operation ID:
{disableOpId}");
            }
        }
    }
}

Console.WriteLine("\\nThis concludes the example scenario.");
Console.WriteLine("Thanks for watching!");
Console.WriteLine(new string('-', 88));
}
catch (Exception ex)
{
    logger.LogError(ex, "An error occurred during the Control Tower
scenario.");
    Console.WriteLine($"An error occurred: {ex.Message}");
}
}

/// <summary>
/// Sets up AWS Organizations and creates or finds a Sandbox OU.
/// </summary>
/// <returns>The ARN of the Sandbox organizational unit.</returns>
private static async Task<string> SetupOrganizationAsync()
{
    Console.WriteLine("\\nChecking organization status...");

    try

```

```

    {
        var orgResponse = await orgClient!.DescribeOrganizationAsync(new
DescribeOrganizationRequest());
        var orgId = orgResponse.Organization.Id;
        Console.WriteLine($"Account is part of organization: {orgId}");
    }
    catch (AWSOrganizationsNotInUseException)
    {
        Console.WriteLine("No organization found. Creating a new
organization...");
        var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
        var orgId = createResponse.Organization.Id;
        Console.WriteLine($"Created new organization: {orgId}");
    }

    // Look for Sandbox OU
    var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
    var rootId = roots.Roots[0].Id;

    Console.WriteLine("Checking for Sandbox OU...");
    var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
ListOrganizationalUnitsForParentRequest { ParentId = rootId });
    var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
"Sandbox");

    if (sandboxOu == null)
    {
        Console.WriteLine("Creating Sandbox OU...");
        var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
        sandboxOu = createOuResponse.OrganizationalUnit;
        Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
    }
    else
    {
        Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
    }

    return sandboxOu.Arn;
}

/// <summary>

```

```

    /// Gets user confirmation by waiting for input or returning true if not
    interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
    false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
}

```

Métodos de contenedor que llama el escenario para administrar las acciones de Aurora.

```

using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
    param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
    param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
        IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
    }
}

```

```

        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
                _controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
                _controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

```

```

        await foreach (var response in baselinesPaginator.Responses)
        {
            baselines.AddRange(response.Baselines);
        }

        return baselines;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// List all enabled baselines.
/// </summary>
/// <returns>A list of enabled baseline summaries.</returns>
public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
{
    try
    {
        var enabledBaselines = new List<EnabledBaselineSummary>();

        var enabledBaselinesPaginator =
            _controlTowerService.Paginators.ListEnabledBaselines(new
            ListEnabledBaselinesRequest());

        await foreach (var response in enabledBaselinesPaginator.Responses)
        {
            enabledBaselines.AddRange(response.EnabledBaselines);
        }

        return enabledBaselines;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't list enabled baselines. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

```

    }

    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
            }
        }
    }

```

```

        if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
        {
            break;
        }
        await Task.Delay(30000); // Wait 30 seconds
    }

    return response.Arn;
}
catch (ValidationException ex)
{
    if (ex.Message.Contains("already enabled"))
        Console.WriteLine("Baseline is already enabled for this target");
    else { Console.WriteLine(ex.Message); }
    // Write the message and return null if baseline cannot be enabled.
    return null;
}
catch (AmazonControlTowerException ex)
{
    Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
    throw;
}
}

}

/// <summary>
/// Disable a baseline for a specific target and wait for the operation to
complete.
/// </summary>
/// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
/// <returns>The operation ID or null if there was a conflict.</returns>
public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
{
    try
    {
        var request = new DisableBaselineRequest
        {
            EnabledBaselineIdentifier = enabledBaselineIdentifier
        };
    }
}

```



```

        var response = await _controlTowerService.DisableBaselineAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetBaselineOperationAsync(operationId);
            Console.WriteLine($"Baseline operation status: {status}");
            if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (ConflictException ex)
    {
        Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
        return null;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Reset an enabled baseline for a specific target.
/// </summary>
/// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
/// <returns>The operation ID.</returns>
public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
{
    try
    {

```

```

        var request = new ResetEnabledBaselineRequest
        {
            EnabledBaselineIdentifier = enabledBaselineIdentifier
        };

        var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetBaselineOperationAsync(operationId);
            Console.WriteLine($"Baseline operation status: {status}");
            if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException)
    {
        Console.WriteLine("Target not found, unable to reset enabled
baseline.");
        throw;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Get the status of a baseline operation.
/// </summary>
/// <param name="operationId">The ID of the baseline operation.</param>
/// <returns>The operation status.</returns>

```

```

    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

```

```

        var enabledControls = new List<EnabledControlSummary>();

        var enabledControlsPaginator =
            _controlTowerService.Paginators.ListEnabledControls(request);

        await foreach (var response in enabledControlsPaginator.Responses)
        {
            enabledControls.AddRange(response.EnabledControls);
        }

        return enabledControls;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
        (ex.Message.Contains("not registered with AWS Control Tower"))
    {
        Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
        return new List<EnabledControlSummary>();
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Enable a control for a specified target.
/// </summary>
/// <param name="controlArn">The ARN of the control to enable.</param>
/// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
/// <returns>The operation ID or null if already enabled.</returns>
public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
{
    try
    {
        Console.WriteLine(controlArn);
        Console.WriteLine(targetIdentifier);

        var request = new EnableControlRequest

```

```

        {
            ControlIdentifier = controlArn,
            TargetIdentifier = targetIdentifier
        };

        var response = await _controlTowerService.EnableControlAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetControlOperationAsync(operationId);
            Console.WriteLine($"Control operation status: {status}");
            if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
    {
        Console.WriteLine("Control is already enabled for this target");
        return null;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
    {
        Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
        return null;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
        throw;
    }
}

```

```

    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
    ARN).</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> DisableControlAsync(string controlArn, string
    targetIdentifier)
    {
        try
        {
            var request = new DisableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.DisableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
    ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {

```

```

        Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Get the status of a control operation.
/// </summary>
/// <param name="operationId">The ID of the control operation.</param>
/// <returns>The operation status.</returns>
public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
{
    try
    {
        var request = new GetControlOperationRequest
        {
            OperationIdentifier = operationId
        };

        var response = await
_controlTowerService.GetControlOperationAsync(request);
        return response.ControlOperation.Status;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException)
    {
        Console.WriteLine("Operation not found.");
        throw;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

/// <summary>
/// List all controls in the Control Tower control catalog.
/// </summary>
/// <returns>A list of control summaries.</returns>

```

```
public async Task<List<ControlSummary>> ListControlsAsync()
{
    try
    {
        var controls = new List<ControlSummary>();

        var controlsPaginator =
            _controlCatalogService.Paginators.ListControls(new
            Amazon.ControlCatalog.Model.ListControlsRequest());

        await foreach (var response in controlsPaginator.Responses)
        {
            controls.AddRange(response.Controls);
        }

        return controls;
    }
    catch (AmazonControlCatalogException ex)
    {
        Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
{ex.Message}");
        throw;
    }
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para .NET .

- [CreateLandingZone](#)
- [DeleteLandingZone](#)
- [DisableBaseline](#)
- [DisableControl](#)
- [EnableBaseline](#)
- [EnableControl](#)
- [GetControlOperation](#)
- [GetLandingZoneOperation](#)

- [ListBaselines](#)
- [ListEnabledBaselines](#)
- [ListEnabledControls](#)
- [ListLandingZones](#)
- [ResetEnabledBaseline](#)

Acciones

DisableBaseline

En el siguiente ejemplo de código, se muestra cómo utilizar DisableBaseline.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Disable a baseline for a specific target and wait for the operation to
complete.
/// </summary>
/// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
/// <returns>The operation ID or null if there was a conflict.</returns>
public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
{
    try
    {
        var request = new DisableBaselineRequest
        {
            EnabledBaselineIdentifier = enabledBaselineIdentifier
        };

        var response = await _controlTowerService.DisableBaselineAsync(request);
        var operationId = response.OperationIdentifier;
```

```

        // Wait for operation to complete
        while (true)
        {
            var status = await GetBaselineOperationAsync(operationId);
            Console.WriteLine($"Baseline operation status: {status}");
            if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (ConflictException ex)
    {
        Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
        return null;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [DisableBaseline](#) la Referencia AWS SDK para .NET de la API.

DisableControl

En el siguiente ejemplo de código, se muestra cómo utilizar `DisableControl`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Disable a control for a specified target.
/// </summary>
/// <param name="controlArn">The ARN of the control to disable.</param>
/// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
/// <returns>The operation ID.</returns>
public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
{
    try
    {
        var request = new DisableControlRequest
        {
            ControlIdentifier = controlArn,
            TargetIdentifier = targetIdentifier
        };

        var response = await _controlTowerService.DisableControlAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetControlOperationAsync(operationId);
            Console.WriteLine($"Control operation status: {status}");
            if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }
    }
}
```

```

        return operationId;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException)
    {
        Console.WriteLine("Control not found.");
        throw;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [DisableControl](#) la Referencia AWS SDK para .NET de la API.

EnableBaseline

En el siguiente ejemplo de código, se muestra cómo utilizar EnableBaseline.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Enable a baseline for the specified target.
/// </summary>
/// <param name="targetIdentifier">The ARN of the target.</param>
/// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
/// <param name="baselineVersion">The version of baseline to enable.</param>
/// <param name="identityCenterBaseline">The identifier of identity center
baseline if it is enabled.</param>

```

```

    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {

```

```

        if (ex.Message.Contains("already enabled"))
            Console.WriteLine("Baseline is already enabled for this target");
        else { Console.WriteLine(ex.Message); }
        // Write the message and return null if baseline cannot be enabled.
        return null;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [EnableBaseline](#) la Referencia AWS SDK para .NET de la API.

EnableControl

En el siguiente ejemplo de código, se muestra cómo utilizar `EnableControl`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Enable a control for a specified target.
/// </summary>
/// <param name="controlArn">The ARN of the control to enable.</param>
/// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
/// <returns>The operation ID or null if already enabled.</returns>
public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
{

```

```

    try
    {
        Console.WriteLine(controlArn);
        Console.WriteLine(targetIdentifier);

        var request = new EnableControlRequest
        {
            ControlIdentifier = controlArn,
            TargetIdentifier = targetIdentifier
        };

        var response = await _controlTowerService.EnableControlAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetControlOperationAsync(operationId);
            Console.WriteLine($"Control operation status: {status}");
            if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
    {
        Console.WriteLine("Control is already enabled for this target");
        return null;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
    {
        Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
        return null;
    }
    catch (AmazonControlTowerException ex)
    {

```

```
        Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:  
{ex.Message}");  
        throw;  
    }  
}
```

- Para obtener más información sobre la API, consulta [EnableControl](#) la Referencia AWS SDK para .NET de la API.

GetBaselineOperation

En el siguiente ejemplo de código, se muestra cómo utilizar `GetBaselineOperation`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// Get the status of a baseline operation.  
/// </summary>  
/// <param name="operationId">The ID of the baseline operation.</param>  
/// <returns>The operation status.</returns>  
public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string  
operationId)  
{  
    try  
    {  
        var request = new GetBaselineOperationRequest  
        {  
            OperationIdentifier = operationId  
        };  
  
        var response = await  
_controlTowerService.GetBaselineOperationAsync(request);  
        return response.BaselineOperation.Status;  
    }  
}
```



```

    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException)
    {
        Console.WriteLine("Operation not found.");
        throw;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [GetBaselineOperation](#) la Referencia AWS SDK para .NET de la API.

GetControlOperation

En el siguiente ejemplo de código, se muestra cómo utilizar `GetControlOperation`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Get the status of a control operation.
/// </summary>
/// <param name="operationId">The ID of the control operation.</param>
/// <returns>The operation status.</returns>
public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
{
    try
    {

```

```
var request = new GetControlOperationRequest
{
    OperationIdentifier = operationId
};

var response = await
_controlTowerService.GetControlOperationAsync(request);
return response.ControlOperation.Status;
}
catch (Amazon.ControlTower.Model.ResourceNotFoundException)
{
    Console.WriteLine("Operation not found.");
    throw;
}
catch (AmazonControlTowerException ex)
{
    Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
    throw;
}
}
```

- Para obtener más información sobre la API, consulta [GetControlOperation](#) la Referencia AWS SDK para .NET de la API.

ListBaselines

En el siguiente ejemplo de código, se muestra cómo utilizar ListBaselines.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// List all baselines.
```

```
/// </summary>
/// <returns>A list of baseline summaries.</returns>
public async Task<List<BaselineSummary>> ListBaselinesAsync()
{
    try
    {
        var baselines = new List<BaselineSummary>();

        var baselinesPaginator =
            _controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

        await foreach (var response in baselinesPaginator.Responses)
        {
            baselines.AddRange(response.Baselines);
        }

        return baselines;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}: {ex.Message}");
        throw;
    }
}
```

- Para obtener más información sobre la API, consulta [ListBaselines](#) la Referencia AWS SDK para .NET de la API.

ListEnabledBaselines

En el siguiente ejemplo de código, se muestra cómo utilizar ListEnabledBaselines.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
                _controlTowerService.Paginators.ListEnabledBaselines(new
                    ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
                {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [ListEnabledBaselines](#) la Referencia AWS SDK para .NET de la API.

ListEnabledControls

En el siguiente ejemplo de código, se muestra cómo utilizar ListEnabledControls.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// List enabled controls for a target organizational unit.
/// </summary>
/// <param name="targetIdentifier">The target organizational unit identifier.</
param>
/// <returns>A list of enabled control summaries.</returns>
public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
targetIdentifier)
{
    try
    {
        var request = new ListEnabledControlsRequest
        {
            TargetIdentifier = targetIdentifier
        };

        var enabledControls = new List<EnabledControlSummary>();

        var enabledControlsPaginator =
_controlTowerService.Paginators.ListEnabledControls(request);

        await foreach (var response in enabledControlsPaginator.Responses)
        {
            enabledControls.AddRange(response.EnabledControls);
        }

        return enabledControls;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
    {
        Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
        return new List<EnabledControlSummary>();
    }
}
```

```
    }  
    catch (AmazonControlTowerException ex)  
    {  
        Console.WriteLine($"Couldn't list enabled controls. Here's why:  
{ex.ErrorCode}: {ex.Message}");  
        throw;  
    }  
}
```

- Para obtener más información sobre la API, consulta [ListEnabledControls](#) la Referencia AWS SDK para .NET de la API.

ListLandingZones

En el siguiente ejemplo de código, se muestra cómo utilizar ListLandingZones.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// List the AWS Control Tower landing zones for an account.  
/// </summary>  
/// <returns>A list of LandingZoneSummary objects.</returns>  
public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()  
{  
    try  
    {  
        var landingZones = new List<LandingZoneSummary>();  
  
        var landingZonesPaginator =  
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());  
  
        await foreach (var response in landingZonesPaginator.Responses)  
        {
```

```

        landingZones.AddRange(response.LandingZones);
    }

    return landingZones;
}
catch (AmazonControlTowerException ex)
{
    Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
    throw;
}
}

```

- Para obtener más información sobre la API, consulta [ListLandingZones](#) la Referencia AWS SDK para .NET de la API.

ResetEnabledBaseline

En el siguiente ejemplo de código, se muestra cómo utilizar ResetEnabledBaseline.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Reset an enabled baseline for a specific target.
/// </summary>
/// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
/// <returns>The operation ID.</returns>
public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
{
    try
    {

```

```

        var request = new ResetEnabledBaselineRequest
        {
            EnabledBaselineIdentifier = enabledBaselineIdentifier
        };

        var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
        var operationId = response.OperationIdentifier;

        // Wait for operation to complete
        while (true)
        {
            var status = await GetBaselineOperationAsync(operationId);
            Console.WriteLine($"Baseline operation status: {status}");
            if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
            {
                break;
            }
            await Task.Delay(30000); // Wait 30 seconds
        }

        return operationId;
    }
    catch (Amazon.ControlTower.Model.ResourceNotFoundException)
    {
        Console.WriteLine("Target not found, unable to reset enabled
baseline.");
        throw;
    }
    catch (AmazonControlTowerException ex)
    {
        Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [ResetEnabledBaseline](#) la Referencia AWS SDK para .NET de la API.

Ejemplos SDK para .NET de DynamoDB con (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con DynamoDB.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

Introducción a DynamoDB

En el siguiente ejemplo de código, se muestra cómo empezar a utilizar DynamoDB.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.Model;  
using Microsoft.Extensions.DependencyInjection;
```

```

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
        Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                             "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
    }
}

```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
{ex.Message}");
        }
    }
}
```

- Para obtener más información sobre la API, consulta [ListTables](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Creación de una tabla que pueda contener datos de películas.
- Colocar, obtener y actualizar una sola película en la tabla.
- Escribir los datos de películas en la tabla a partir de un archivo JSON de ejemplo.
- Consultar películas que se hayan estrenado en un año determinado.
- Buscar películas que se hayan estrenado en un intervalo de años.
- Eliminación de una película de la tabla y, a continuación, eliminar la tabla.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
            Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
                .ConfigureServices((_, services) =>
                    services.AddAWSService<IAmazonDynamoDB>()
                        .AddTransient<DynamoDbWrapper>())
                .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";
    }
}

```

```
DisplayInstructions();

// Create a new table and wait for it to be active.
Console.WriteLine($"Creating the new table: {tableName}");

var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

Console.WriteLine(success
    ? $"\\nTable: {tableName} successfully created."
    : $"\\nCould not create {tableName}.");

WaitForEnter();

// Add a single new movie to the table.
var newMovie = new Movie
{
    Year = 2021,
    Title = "Spider-Man: No Way Home",
};

success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
if (success)
{
    Console.WriteLine($"Added {newMovie.Title} to the table.");
}
else
{
    Console.WriteLine("Could not add movie to table.");
}

WaitForEnter();

// Update the new movie by adding a plot and rank.
var newInfo = new MovieInfo
{
    Plot = "With Spider-Man's identity now revealed, Peter asks" +
        "Doctor Strange for help. When a spell goes wrong, dangerous" +
        "foes from other worlds start to appear, forcing Peter to" +
        "discover what it truly means to be Spider-Man.",
    Rank = 9,
};

success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
```

```
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
```

```
};

success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

if (success)
{
    Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
}
else
{
    Console.WriteLine($"Could not delete {movieToDelete.Title}.");
}

WaitForEnter();

// Use Query to find all the movies released in 2010.
int findYear = 2010;
Console.WriteLine($"Movies released in {findYear}");
var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
Console.WriteLine($"Found {queryCount} movies released in {findYear}");

WaitForEnter();

// Use Scan to get a list of movies from 2001 to 2011.
int startYear = 2001;
int endYear = 2011;
var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

WaitForEnter();

// Delete the table.
success = await dynamoDbWrapper.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}
```

```

    }

    Console.WriteLine("The DynamoDB Basics example application is complete.");

    WaitForEnter();
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
private static void DisplayInstructions()
{
    if (!IsInteractive)
    {
        return;
    }

    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 28));
    Console.WriteLine("DynamoDB Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using DynamoDB
with the AWS SDK.");
    Console.WriteLine(SepBar);
    Console.WriteLine("The application does the following:");
    Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
    Console.WriteLine("\t2. Adds a single movie to the table.");
    Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
    Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
    Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
    Console.WriteLine("\t6. Deletes a movie.");
    Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
given year.");
    Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
a range of years.");
    Console.WriteLine("\t9. Finally, it deletes the table that was just
created.");
    WaitForEnter();
}

/// <summary>

```



```

    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}

```

Utilice el cliente inyectado para las operaciones de tabla.

```

using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
}

```

Crea una tabla para contener los datos de las películas.

```

    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

```

```
// Wait until the table is ACTIVE and then report success.
Console.WriteLine("Waiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = response.TableDescription.TableName,
};

TableStatus status;

int sleepDuration = 2000;

do
{
    Thread.Sleep(sleepDuration);

    var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException ex)
{
    Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
    throw;
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
    throw;
}
```

```
}
```

Agrega una sola película a la tabla.

```

    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
    added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {

```

```

        Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
        throw;
    }
}

```

Actualiza un solo elemento de una tabla.

```

/// <summary>
/// Updates an existing item in the movies table.
/// </summary>
/// <param name="newMovie">A Movie object containing information for
/// the movie to update.</param>
/// <param name="newInfo">A MovieInfo object that contains the
/// information that will be changed.</param>
/// <param name="tableName">The name of the table that contains the movie.</
param>
/// <returns>A Boolean value that indicates the success of the operation.</
returns>
public async Task<bool> UpdateItemAsync(
    Movie newMovie,
    MovieInfo newInfo,
    string tableName)
{
    try
    {
        var key = new Dictionary<string, AttributeValue>
        {
            ["title"] = new AttributeValue { S = newMovie.Title },
            ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
        };
        var updates = new Dictionary<string, AttributeValueUpdate>
        {
            ["info.plot"] = new AttributeValueUpdate

```

```
        {
            Action = AttributeAction.PUT,
            Value = new AttributeValue { S = newInfo.Plot },
        },

        ["info.rating"] = new AttributeValueUpdate
        {
            Action = AttributeAction.PUT,
            Value = new AttributeValue { N = newInfo.Rank.ToString() },
        },
    };

    var request = new UpdateItemRequest
    {
        AttributeUpdates = updates,
        Key = key,
        TableName = tableName,
    };

    await _amazonDynamoDB.UpdateItemAsync(request);
    return true;
}
catch (ResourceNotFoundException ex)
{
    Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
    return false;
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
    throw;
}
}
```

Recupera un solo elemento de la tabla de películas.

```

/// <summary>
/// Gets information about an existing movie from the table.
/// </summary>
/// <param name="newMovie">A Movie object containing information about
/// the movie to retrieve.</param>
/// <param name="tableName">The name of the table containing the movie.</param>
/// <returns>A Dictionary object containing information about the item
/// retrieved.</returns>
public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
{
    try
    {
        var key = new Dictionary<string, AttributeValue>
        {
            ["title"] = new AttributeValue { S = newMovie.Title },
            ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
        };

        var request = new GetItemRequest
        {
            Key = key,
            TableName = tableName,
        };

        var response = await _amazonDynamoDB.GetItemAsync(request);
        return response.Item;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
        return new Dictionary<string, AttributeValue>();
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {

```

```

        Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
        throw;
    }
}

```

Escribe un lote de elementos en la tabla de películas.

```

/// <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 name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>

```



```
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public async Task<long> BatchWriteItemsAsync(
    string movieFileName, string tableName)
{
    try
    {
        var movies = ImportMovies(movieFileName);
        if (!movies.Any())
        {
            Console.WriteLine("Couldn't find the JSON file with movie data.");
            return 0;
        }

        var context = new DynamoDBContextBuilder()
            // Optional call to provide a specific instance of IAmazonDynamoDB
            .WithDynamoDBClient(() => _amazonDynamoDB)
            .Build();

        var movieBatch = context.CreateBatchWrite<Movie>(
            new BatchWriteConfig()
            {
                OverrideTableName = tableName
            });
        movieBatch.AddPutItems(movies);

        Console.WriteLine("Adding imported movies to the table.");
        await movieBatch.ExecuteAsync();

        return movies.Count;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
        throw;
    }
    catch (AmazonDynamoDBException ex)
    {
    }
```

```

        Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
        throw;
    }
}

```

Elimina un solo elemento de la tabla.

```

/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="tableName">The name of the table from which the item
/// will be deleted.</param>
/// <param name="movieToDelete">A movie object containing the title and
/// year of the movie to delete.</param>
/// <returns>A Boolean value indicating the success or failure of the
/// delete operation.</returns>
public async Task<bool> DeleteItemAsync(
    string tableName,
    Movie movieToDelete)
{
    try
    {
        var key = new Dictionary<string, AttributeValue>
        {
            ["title"] = new AttributeValue { S = movieToDelete.Title },
            ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
        };

        var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

        await _amazonDynamoDB.DeleteItemAsync(request);
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred during delete
operation. {ex.Message}");
        throw;
    }
}

```

```

    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
        return false;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
        throw;
    }
}

```

Consulta en la tabla las películas estrenadas en un año determinado.

```

/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);
    }
}

```

```
Console.WriteLine("\nFind movies released in: {year}:");

var config = new QueryOperationConfig()
{
    Limit = 10, // 10 items per page.
    Select = SelectValues.SpecificAttributes,
    AttributesToGet = new List<string>
    {
        "title",
        "year",
    },
    ConsistentRead = true,
    Filter = filter,
};

// Value used to track how many movies match the
// supplied criteria.
var moviesFound = 0;

var search = movieTable.Query(config);
do
{
    var movieList = await search.GetNextSetAsync();
    moviesFound += movieList.Count;

    foreach (var movie in movieList)
    {
        DisplayDocument(movie);
    }
}
while (!search.IsDone);

return moviesFound;
}
catch (ResourceNotFoundException ex)
{
    Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
    return 0;
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
}
```

```

        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
        throw;
    }
}

```

Busca en la tabla películas que se hayan estrenado en un intervalo de años.

```

/// <summary>
/// Scans the table for movies released between the specified years.
/// </summary>
/// <param name="tableName">The name of the table to scan.</param>
/// <param name="startYear">The starting year for the range.</param>
/// <param name="endYear">The ending year for the range.</param>
/// <returns>The number of movies found in the specified year range.</returns>
public async Task<int> ScanTableAsync(
    string tableName,
    int startYear,
    int endYear)
{
    try
    {
        var request = new ScanRequest
        {
            TableName = tableName,
            ExpressionAttributeNames = new Dictionary<string, string>
            {
                { "#yr", "year" },
            },
            ExpressionAttributeValues = new Dictionary<string, AttributeValue>
            {
                { ":y_a", new AttributeValue { N = startYear.ToString() } },
                { ":y_z", new AttributeValue { N = endYear.ToString() } },
            },
            FilterExpression = "#yr between :y_a and :y_z",
            ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",

```

```

        Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
    };

    // Keep track of how many movies were found.
    int foundCount = 0;

    var response = new ScanResponse();
    do
    {
        response = await _amazonDynamoDB.ScanAsync(request);
        foundCount += response.Items.Count;
        response.Items.ForEach(i => DisplayItem(i));
        request.ExclusiveStartKey = response.LastEvaluatedKey;
    }
    while (response?.LastEvaluatedKey?.Count > 0);
    return foundCount;
}
catch (ResourceNotFoundException ex)
{
    Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
    return 0;
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
    throw;
}
}

```

Elimina la tabla de películas.

```

/// <summary>
/// Deletes a DynamoDB table.
/// </summary>

```

```

/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the operation.</returns>
public async Task<bool> DeleteTableAsync(string tableName)
{
    try
    {
        var request = new DeleteTableRequest
        {
            TableName = tableName,
        };

        var response = await _amazonDynamoDB.DeleteTableAsync(request);

        Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
        return true;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
        return false;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
        return false;
    }
}

```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para .NET .
 - [BatchWriteItem](#)

- [CreateTable](#)
- [DeleteItem](#)
- [DeleteTable](#)
- [DescribeTable](#)
- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

Acciones

BatchWriteItem

En el siguiente ejemplo de código, se muestra cómo utilizar BatchWriteItem.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Escribe un lote de elementos en la tabla de películas.

```
/// <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 name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
```



```

        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public async Task<long> BatchWriteItemsAsync(
    string movieFileName, string tableName)
{
    try
    {
        var movies = ImportMovies(movieFileName);
        if (!movies.Any())
        {
            Console.WriteLine("Couldn't find the JSON file with movie data.");
            return 0;
        }

        var context = new DynamoDBContextBuilder()
            // Optional call to provide a specific instance of IAmazonDynamoDB
            .WithDynamoDBClient(() => _amazonDynamoDB)
            .Build();
    }
}

```

```

        var movieBatch = context.CreateBatchWrite<Movie>(
            new BatchWriteConfig()
            {
                OverrideTableName = tableName
            });
        movieBatch.AddPutItems(movies);

        Console.WriteLine("Adding imported movies to the table.");
        await movieBatch.ExecuteAsync();

        return movies.Count;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
        throw;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [BatchWriteItem](#) la Referencia AWS SDK para .NET de la API.

CreateTable

En el siguiente ejemplo de código, se muestra cómo utilizar CreateTable.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Creates a new Amazon DynamoDB table and then waits for the new
/// table to become active.
/// </summary>
/// <param name="tableName">The name of the table to create.</param>
/// <returns>A Boolean value indicating the success of the operation.</returns>
public async Task<bool> CreateMovieTableAsync(string tableName)
{
    try
    {
        var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition
                {
                    AttributeName = "title",
                    AttributeType = ScalarAttributeType.S,
                },
                new AttributeDefinition
                {
                    AttributeName = "year",
                    AttributeType = ScalarAttributeType.N,
                },
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement
                {
                    AttributeName = "year",
                    KeyType = KeyType.HASH,
```

```

        },
        new KeySchemaElement
        {
            AttributeName = "title",
            KeyType = KeyType.RANGE,
        },
    },
    BillingMode = BillingMode.PAY_PER_REQUEST,
});

// Wait until the table is ACTIVE and then report success.
Console.WriteLine("Waiting for table to become active...");

var request = new DescribeTableRequest
{
    TableName = response.TableDescription.TableName,
};

TableStatus status;

int sleepDuration = 2000;

do
{
    Thread.Sleep(sleepDuration);

    var describeTableResponse = await
        _amazonDynamoDB.DescribeTableAsync(request);
    status = describeTableResponse.Table.TableStatus;

    Console.WriteLine(".");
}
while (status != "ACTIVE");

return status == TableStatus.ACTIVE;
}
catch (ResourceInUseException ex)
{
    Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
    throw;
}
catch (AmazonDynamoDBException ex)
{

```

```
        Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
        throw;
    }
}
```

- Para obtener más información sobre la API, consulta [CreateTable](#) la Referencia AWS SDK para .NET de la API.

DeleteItem

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteItem.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="tableName">The name of the table from which the item
/// will be deleted.</param>
/// <param name="movieToDelete">A movie object containing the title and
/// year of the movie to delete.</param>
/// <returns>A Boolean value indicating the success or failure of the
/// delete operation.</returns>
public async Task<bool> DeleteItemAsync(
    string tableName,
```

```

        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [DeleteItem](#) la Referencia AWS SDK para .NET de la API.

DeleteTable

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteTable.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Deletes a DynamoDB table.
/// </summary>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the operation.</returns>
public async Task<bool> DeleteTableAsync(string tableName)
{
    try
    {
        var request = new DeleteTableRequest
        {
            TableName = tableName,
        };

        var response = await _amazonDynamoDB.DeleteTableAsync(request);

        Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
        return true;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
        return false;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
    }
}
```

```

    {
        Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
        return false;
    }
}

```

- Para obtener más información sobre la API, consulta [DeleteTable](#) la Referencia AWS SDK para .NET de la API.

GetItem

En el siguiente ejemplo de código, se muestra cómo utilizar `GetItem`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Gets information about an existing movie from the table.
/// </summary>
/// <param name="newMovie">A Movie object containing information about
/// the movie to retrieve.</param>
/// <param name="tableName">The name of the table containing the movie.</param>
/// <returns>A Dictionary object containing information about the item
/// retrieved.</returns>
public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
{
    try
    {
        var key = new Dictionary<string, AttributeValue>
        {
            ["title"] = new AttributeValue { S = newMovie.Title },

```



```

        ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
    };

    var request = new GetItemRequest
    {
        Key = key,
        TableName = tableName,
    };

    var response = await _amazonDynamoDB.GetItemAsync(request);
    return response.Item;
}
catch (ResourceNotFoundException ex)
{
    Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
    return new Dictionary<string, AttributeValue>();
}
catch (AmazonDynamoDBException ex)
{
    Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
    throw;
}
}

```

- Para obtener más información sobre la API, consulta [GetItem](#) la Referencia AWS SDK para .NET de la API.

PutItem

En el siguiente ejemplo de código, se muestra cómo utilizar PutItem.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Adds a new item to the table.
/// </summary>
/// <param name="newMovie">A Movie object containing information for
/// the movie to add to the table.</param>
/// <param name="tableName">The name of the table where the item will be
added.</param>
/// <returns>A Boolean value that indicates the results of adding the item.</
returns>
public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
{
    try
    {
        var item = new Dictionary<string, AttributeValue>
        {
            ["title"] = new AttributeValue { S = newMovie.Title },
            ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
        };

        var request = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
        };

        await _amazonDynamoDB.PutItemAsync(request);
        return true;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
        return false;
    }
}
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
}
```

- Para obtener más información sobre la API, consulta [PutItem](#) la Referencia AWS SDK para .NET de la API.

Query

En el siguiente ejemplo de código, se muestra cómo utilizar Query.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
```

```
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };

        // Value used to track how many movies match the
        // supplied criteria.
        var moviesFound = 0;

        var search = movieTable.Query(config);
        do
        {
            var movieList = await search.GetNextSetAsync();
            moviesFound += movieList.Count;

            foreach (var movie in movieList)
            {
                DisplayDocument(movie);
            }
        }
        while (!search.IsDone);

        return moviesFound;
    }
}
```

```
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies. {ex.Message}");
            throw;
        }
    }
}
```

- Para obtener información sobre la API, consulte [Query](#) en la referencia de la API de AWS SDK para .NET .

Scan

En el siguiente ejemplo de código, se muestra cómo utilizar Scan.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Scans the table for movies released between the specified years.
/// </summary>
/// <param name="tableName">The name of the table to scan.</param>
/// <param name="startYear">The starting year for the range.</param>
```

```

/// <param name="endYear">The ending year for the range.</param>
/// <returns>The number of movies found in the specified year range.</returns>
public async Task<int> ScanTableAsync(
    string tableName,
    int startYear,
    int endYear)
{
    try
    {
        var request = new ScanRequest
        {
            TableName = tableName,
            ExpressionAttributeNames = new Dictionary<string, string>
            {
                { "#yr", "year" },
            },
            ExpressionAttributeValues = new Dictionary<string, AttributeValue>
            {
                { ":y_a", new AttributeValue { N = startYear.ToString() } },
                { ":y_z", new AttributeValue { N = endYear.ToString() } },
            },
            FilterExpression = "#yr between :y_a and :y_z",
            ProjectionExpression = "#yr, title, info.actors[0], info.directors, info.running_time_secs",
            Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
        };

        // Keep track of how many movies were found.
        int foundCount = 0;

        var response = new ScanResponse();
        do
        {
            response = await _amazonDynamoDB.ScanAsync(request);
            foundCount += response.Items.Count;
            response.Items.ForEach(i => DisplayItem(i));
            request.ExclusiveStartKey = response.LastEvaluatedKey;
        }
        while (response?.LastEvaluatedKey?.Count > 0);
        return foundCount;
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
    }
}

```

```
        return 0;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
        throw;
    }
}
```

- Para obtener información sobre la API, consulte [Scan](#) en la referencia de la API de AWS SDK para .NET .

UpdateItem

En el siguiente ejemplo de código, se muestra cómo utilizar UpdateItem.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Updates an existing item in the movies table.
/// </summary>
/// <param name="newMovie">A Movie object containing information for
/// the movie to update.</param>
/// <param name="newInfo">A MovieInfo object that contains the
/// information that will be changed.</param>
```

```
/// <param name="tableName">The name of the table that contains the movie.</  
param>  
/// <returns>A Boolean value that indicates the success of the operation.</  
returns>  
public async Task<bool> UpdateItemAsync(  
    Movie newMovie,  
    MovieInfo newInfo,  
    string tableName)  
{  
    try  
    {  
        var key = new Dictionary<string, AttributeValue>  
        {  
            ["title"] = new AttributeValue { S = newMovie.Title },  
            ["year"] = new AttributeValue { N = newMovie.Year.ToString() },  
        };  
        var updates = new Dictionary<string, AttributeValueUpdate>  
        {  
            ["info.plot"] = new AttributeValueUpdate  
            {  
                Action = AttributeAction.PUT,  
                Value = new AttributeValue { S = newInfo.Plot },  
            },  
  
            ["info.rating"] = new AttributeValueUpdate  
            {  
                Action = AttributeAction.PUT,  
                Value = new AttributeValue { N = newInfo.Rank.ToString() },  
            },  
        };  
        var request = new UpdateItemRequest  
        {  
            AttributeUpdates = updates,  
            Key = key,  
            TableName = tableName,  
        };  
  
        await _amazonDynamoDB.UpdateItemAsync(request);  
        return true;  
    }  
    catch (ResourceNotFoundException ex)  
    {
```



```
        Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
        return false;
    }
    catch (AmazonDynamoDBException ex)
    {
        Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
        throw;
    }
}
```

- Para obtener más información sobre la API, consulta [UpdateItem](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon EC2 con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon EC2.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)

Introducción

Introducción a Amazon EC2

El siguiente ejemplo de código muestra cómo empezar a utilizar Amazon EC2.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
namespace EC2Actions;

public class HelloEc2
{
    /// <summary>
    /// HelloEc2 lists the existing security groups for the default users.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
        EC2).
        using var host =
            Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
                .ConfigureServices((_, services) =>
                    services.AddAWSService<IAmazonEC2>()
                        .AddTransient<EC2Wrapper>()
                )
                .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);
```

```
        await foreach (var securityGroup in
paginatorForSecurityGroups.SecurityGroups)
        {
            securityGroups.Add(securityGroup);
        }

// Now print the security groups returned by the call to
// DescribeSecurityGroupsAsync.
Console.WriteLine("Welcome to the EC2 Hello Service example. " +
                    "\nLet's list your Security Groups:");
securityGroups.ForEach(group =>
{
    Console.WriteLine(
        $"Security group: {group.GroupName} ID: {group.GroupId}");
});
}
catch (AmazonEC2Exception ex)
{
    Console.WriteLine($"An Amazon EC2 service error occurred while listing
security groups. {ex.Message}");
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while listing security groups.
{ex.Message}");
}
}
```

- Para obtener más información sobre la API, consulta [DescribeSecurityGroups](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon ECS con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon ECS.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)

Introducción

Introducción a Amazon ECS

En el siguiente ejemplo de código se muestra cómo empezar a utilizar Amazon ECS.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        the Amazon ECS client.
        // 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<IAmazonECS>()
        )
        .Build();

var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();

Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
available in the your account");
Console.WriteLine();

var clusters = new List<string>();

var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
ListClustersRequest());

await foreach (var response in clustersPaginator.Responses)
{
    clusters.AddRange(response.ClusterArns);
}

if (clusters.Count > 0)
{
    clusters.ForEach(cluster =>
    {
        Console.WriteLine($"\\tARN: {cluster}");
        Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
        Console.WriteLine();
    });
}
else
{
    Console.WriteLine("No clusters were found.");
}
}
}

```

- Para obtener más información sobre la API, consulta [ListClusters](#) la Referencia AWS SDK para .NET de la API.

AWS IoT ejemplos que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con AWS IoT.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

¿Hola AWS IoT

En el siguiente ejemplo de código se muestra cómo empezar a utilizar AWS IoT.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
```

```
/// <summary>
/// Main method to run the Hello IoT example.
/// </summary>
/// <param name="args">Command line arguments.</param>
/// <returns>A Task object.</returns>
public static async Task Main(string[] args)
{
    var iotClient = new AmazonIoTClient();

    try
    {
        Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
        Console.WriteLine(new string('-', 80));

        // Use pages of 10.
        var request = new ListThingsRequest()
        {
            MaxResults = 10
        };
        var response = await iotClient.ListThingsAsync(request);

        // Since there is not a built-in paginator, use the NextMarker to
paginate.
        bool hasMoreResults = true;

        var things = new List<ThingAttribute>();
        while (hasMoreResults)
        {
            things.AddRange(response.Things);

            // If NextMarker is not null, there are more results. Get the next
page of results.
            if (!String.IsNullOrEmpty(response.NextMarker))
            {
                request.Marker = response.NextMarker;
                response = await iotClient.ListThingsAsync(request);
            }
            else
                hasMoreResults = false;
        }

        if (things is { Count: > 0 })
        {
            Console.WriteLine($"Found {things.Count} IoT Things:");
        }
    }
}
```

```

        foreach (var thing in things)
        {
            Console.WriteLine($"- Thing Name: {thing.ThingName}");
            Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
            Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
            Console.WriteLine($"  Version: {thing.Version}");

            if (thing.Attributes?.Count > 0)
            {
                Console.WriteLine("  Attributes:");
                foreach (var attr in thing.Attributes)
                {
                    Console.WriteLine($"    {attr.Key}: {attr.Value}");
                }
            }
            Console.WriteLine();
        }
    }
    else
    {
        Console.WriteLine("No IoT Things found in your account.");
        Console.WriteLine("You can create IoT Things using the IoT Basics
scenario example.");
    }

    Console.WriteLine("Hello IoT completed successfully.");
}
catch (Amazon.IoT.Model.ThrottlingException ex)
{
    Console.WriteLine($"Request throttled, please try again later:
{ex.Message}");
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
}
}
}

```

- Para obtener información sobre la API, consulte [listThings](#) en la Referencia de la API de AWS SDK para .NET .

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crea una AWS IoT cosa.
- Generar un certificado de dispositivo.
- Actualiza AWS IoT cualquier cosa con atributos.
- Devolver un punto de conexión único.
- Enumere sus AWS IoT certificados.
- Actualiza y AWS IoT oculta.
- Escribir la información de estado.
- Crear una regla.
- Mostrar las reglas.
- Buscar objetos con el nombre de objeto.
- Elimina cualquier AWS IoT cosa.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecute un escenario interactivo que demuestre AWS IoT las funciones.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
```

```

private static IoTWrapper _iotWrapper = null!;
private static IAmazonCloudFormation _amazonCloudFormation = null!;
private static ILogger<IoTBasics> _logger = null!;

private static string _stackName = "IoTBasicsStack";
private static string _stackResourcePath = "../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

/// <summary>
/// Main method for the IoT 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 the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
RegionEndpoint.USEast1 })
            .AddAWSService<IAmazonCloudFormation>()
            .AddTransient<IoTWrapper>()
            .AddLogging(builder => builder.AddConsole())
            .AddSingleton<IAmazonIotData>(sp =>
            {
                var iotService = sp.GetRequiredService<IAmazonIoT>();
                var request = new DescribeEndpointRequest
                {
                    EndpointType = "iot:Data-ATS"
                };
                var response =
                iotService.DescribeEndpointAsync(request).Result;
                return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
            })
        )
        .Build();

    logger = LoggerFactory.Create(builder => builder.AddConsole())
        .CreateLogger<IoTBasics>();

    Wrapper = host.Services.GetRequiredService<IoTWrapper>();
    CloudFormationClient =
    host.Services.GetRequiredService<IAmazonCloudFormation>();

```

```

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

```

```

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);

```

```

        Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
        Console.WriteLine(new string('-', 80));

        // Step 1.1: List AWS IoT Things
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("2. List AWS IoT Things.");
        Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var things = await iotWrapper.ListThingsAsync();
        Console.WriteLine($"Found {things.Count} IoT Things:");
        foreach (var thing in things.Take(10)) // Show first 10 things
        {
            Console.WriteLine($"Thing Name: {thing.ThingName}");
            Console.WriteLine($"Thing ARN: {thing.ThingArn}");
            if (thing.Attributes != null && thing.Attributes.Any())
            {
                Console.WriteLine("Attributes:");
                foreach (var attr in thing.Attributes)
                {
                    Console.WriteLine($"  {attr.Key}: {attr.Value}");
                }
            }
            Console.WriteLine("-----");
        }
        Console.WriteLine();
        Console.WriteLine(new string('-', 80));

        // Step 2: Generate a Device Certificate
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("3. Generate a device certificate.");
        Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
        Console.WriteLine();

        var createCert = "y";
        if (IsInteractive)

```

```

        {
            Console.WriteLine($"Do you want to create a certificate for {thingName}?
(y/n)");
            createCert = Console.ReadLine();
        }
        else
        {
            Console.WriteLine($"Creating certificate for {thingName}...");
        }

        if (createCert?.ToLower() == "y")
        {
            var certificateResult = await
            iotWrapper.CreateKeysAndCertificateAsync();
            if (certificateResult.HasValue)
            {
                var (certArn, certPem, certId) = certificateResult.Value;
                certificateArn = certArn;
                certificateId = certId;

                Console.WriteLine($"\\nCertificate:");
                // Show only first few lines of certificate for brevity
                var lines = certPem.Split('\\n');
                for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                {
                    Console.WriteLine(lines[i]);
                }
                if (lines.Length > 5)
                {
                    Console.WriteLine("...");
                }

                Console.WriteLine($"\\nCertificate ARN:");
                Console.WriteLine(certificateArn);

                // Step 3: Attach the Certificate to the AWS IoT Thing
                Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");

                var attachResult = await
                iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                if (attachResult)
                {
                    Console.WriteLine("Certificate attached to Thing
successfully.");
                }
            }
        }
    }
}

```

```

        }
        else
        {
            Console.WriteLine("Failed to attach certificate to Thing.");
        }

        Console.WriteLine("Thing Details:");
        Console.WriteLine($"Thing Name: {thingName}");
        Console.WriteLine($"Thing ARN: {thingArn}");
    }
    else
    {
        Console.WriteLine("Failed to create certificate.");
    }
}

Console.WriteLine(new string('-', 80));

// Step 4: Update an AWS IoT Thing with Attributes
Console.WriteLine(new string('-', 80));
Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
Console.WriteLine();
if (IsInteractive)
{
    Console.WriteLine("Press Enter to continue...");
    Console.ReadLine();
}

var attributes = new Dictionary<string, string>
{
    { "Location", "Seattle" },
    { "DeviceType", "Sensor" },
    { "Firmware", "1.2.3" }
};

await iotWrapper.UpdateThingAsync(thingName, attributes);
Console.WriteLine("Thing attributes updated successfully.");
Console.WriteLine(new string('-', 80));

// Step 5: Return a unique endpoint specific to the Amazon Web Services
account

```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var endpoint = await iotWrapper.DescribeEndpointAsync();
        if (endpoint != null)
        {
            var subdomain = endpoint.Split('.')[0];
            Console.WriteLine($"Extracted subdomain: {subdomain}");
            Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
        }
        else
        {
            Console.WriteLine("Failed to retrieve endpoint.");
        }
        Console.WriteLine(new string('-', 80));

        // Step 6: List your AWS IoT certificates
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("6. List your AWS IoT certificates");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var certificates = await iotWrapper.ListCertificatesAsync();
        foreach (var cert in certificates.Take(5)) // Show first 5 certificates
        {
            Console.WriteLine($"Cert id: {cert.CertificateId}");
            Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
        }
        Console.WriteLine();
        Console.WriteLine(new string('-', 80));

        // Step 7: Create an IoT shadow
        Console.WriteLine(new string('-', 80));
```



```

        Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var shadowPayload = JsonSerializer.Serialize(new
        {
            state = new
            {
                desired = new
                {
                    temperature = 25,
                    humidity = 50
                }
            }
        });

        await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
        Console.WriteLine("Thing Shadow updated successfully.");
        Console.WriteLine(new string('-', 80));

        // Step 8: Write out the state information, in JSON format
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("8. Write out the state information, in JSON
format.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
        Console.WriteLine($"Received Shadow Data: {shadowData}");
        Console.WriteLine(new string('-', 80));

        // Step 9: Set up resources (SNS topic and IAM role) and create a rule
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("9. Set up resources and create a rule");
        Console.WriteLine();

```

```

        // Deploy CloudFormation stack to create SNS topic and IAM role
        Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

        var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
        if (deployStack)
        {
            if (IsInteractive)
            {
                Console.Write(
                    $"Enter stack resource file path (or press Enter for default
'_{stackResourcePath}'): ");
                var userResourcePath = Console.ReadLine();
                if (!string.IsNullOrEmpty(userResourcePath))
                    _stackResourcePath = userResourcePath;
            }

            _stackName = PromptUserForStackName();

            var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

            if (deploySuccess)
            {
                // Get stack outputs
                var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                if (stackOutputs != null)
                {
                    snsTopicArn = stackOutputs["SNSTopicArn"];
                    string roleArn = stackOutputs["RoleArn"];

                    Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                    Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                    if (IsInteractive)
                    {
                        Console.Write($"Enter Rule name (press Enter for default
'_{ruleName}'): ");
                        var userRuleName = Console.ReadLine();
                        if (!string.IsNullOrEmpty(userRuleName))

```

```

        ruleName = userRuleName;
    }
    else
    {
        Console.WriteLine($"Using default rule name:
{ruleName}");
    }

    // Now create the IoT rule with the CloudFormation outputs
    var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
    if (ruleResult)
    {
        Console.WriteLine("IoT Rule created successfully.");
    }
    else
    {
        Console.WriteLine("Failed to create IoT rule.");
    }
}
else
{
    Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
}
}
else
{
    Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
}
}
else
{
    Console.WriteLine("Skipping CloudFormation stack deployment and rule
creation.");
}
Console.WriteLine(new string('-', 80));

// Step 10: List your rules
Console.WriteLine(new string('-', 80));
Console.WriteLine("10. List your rules.");
if (IsInteractive)
{

```

```

        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine();
    }

    var rules = await iotWrapper.ListTopicRulesAsync();
    Console.WriteLine("List of IoT Rules:");
    foreach (var rule in rules.Take(5)) // Show first 5 rules
    {
        Console.WriteLine($"Rule Name: {rule.RuleName}");
        Console.WriteLine($"Rule ARN: {rule.RuleArn}");
        Console.WriteLine("-----");
    }
    Console.WriteLine();
    Console.WriteLine(new string('-', 80));

    // Step 11: Search things using the Thing name
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("11. Search things using the Thing name.");
    if (IsInteractive)
    {
        Console.WriteLine("Press Enter to continue...");
        Console.ReadLine();
    }

    var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
    if (searchResults.Any())
    {
        Console.WriteLine($"Thing id found using search is
{searchResults.First().ThingId}");
    }
    else
    {
        Console.WriteLine($"No search results found for Thing:
{thingName}");
    }
    Console.WriteLine(new string('-', 80));

    // Step 12: Cleanup - Detach and delete certificate
    if (!string.IsNullOrEmpty(certificateArn))
    {
        Console.WriteLine(new string('-', 80));
        var deleteCert = "y";
        if (IsInteractive)

```

```

        {
            Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
            deleteCert = Console.ReadLine();
        }
        else
        {
            Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
        }

        if (deleteCert?.ToLower() == "y")
        {
            Console.WriteLine("12. You selected to detach and delete the
certificate.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
            Console.WriteLine($"{{certificateArn}} was successfully removed
from {{thingName}}");

            await iotWrapper.DeleteCertificateAsync(certificateId);
            Console.WriteLine($"{{certificateArn}} was successfully
deleted.");
        }
        Console.WriteLine(new string('-', 80));
    }

    // Step 13: Delete the AWS IoT Thing
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("13. Delete the AWS IoT Thing.");
    var deleteThing = "y";
    if (IsInteractive)
    {
        Console.Write($"Do you want to delete the IoT Thing? (y/n)");
        deleteThing = Console.ReadLine();
    }
    else
    {

```

```

        Console.WriteLine($"Deleting IoT Thing {thingName}...");
    }

    if (deleteThing?.ToLower() == "y")
    {
        await iotWrapper.DeleteThingAsync(thingName);
        Console.WriteLine($"Deleted Thing {thingName}");
    }
    Console.WriteLine(new string('-', 80));

    // Step 14: Clean up CloudFormation stack
    if (!string.IsNullOrEmpty(snsTopicArn))
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("14. Clean up CloudFormation stack.");
        Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

        var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
        if (cleanup)
        {
            var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);

            var stackCleanupSuccess = await
DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
            if (ruleCleanupSuccess && stackCleanupSuccess)
            {
                Console.WriteLine("Successfully cleaned up CloudFormation
stack and all resources.");
            }
            else
            {
                Console.WriteLine("Some cleanup operations failed. Check the
logs for details.");
            }
        }
        else
        {
            Console.WriteLine($"Resources will remain. Stack name:
{_stackName}");
        }
        Console.WriteLine(new string('-', 80));
    }

```

```

    }
}
catch (Exception ex)
{
    scenarioLogger.LogError(ex, "Error occurred during scenario
execution.");

    // Cleanup on error
    if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
    {
        try
        {
            await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
            await iotWrapper.DeleteCertificateAsync(certificateId);
        }
        catch (Exception cleanupEx)
        {
            scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
        }
    }

    if (!string.IsNullOrEmpty(thingName))
    {
        try
        {
            await iotWrapper.DeleteThingAsync(thingName);
        }
        catch (Exception cleanupEx)
        {
            scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
        }
    }

    // Clean up CloudFormation stack on error
    if (!string.IsNullOrEmpty(snsTopicArn))
    {
        try
        {
            await _iotWrapper.DeleteTopicRuleAsync(ruleName);
            await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

```

```

        }
        catch (Exception cleanupEx)
        {
            scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
        }
    }

    throw;
}
}

/// <summary>
/// Deploys the CloudFormation stack with the necessary resources.
/// </summary>
/// <param name="stackName">The name of the CloudFormation stack.</param>
/// <param name="cloudFormationClient">The CloudFormation client.</param>
/// <param name="scenarioLogger">The logger.</param>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
{
    Console.WriteLine($"\\nDeploying CloudFormation stack: {stackName}");

    try
    {
        var request = new CreateStackRequest
        {
            StackName = stackName,
            TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
            Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
        };

        var response = await cloudFormationClient.CreateStackAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

            bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

            if (stackCreated)

```



```

        {
            Console.WriteLine("CloudFormation stack created successfully.");
            return true;
        }
        else
        {
            scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
            return false;
        }
    }
    else
    {
        scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
        return false;
    }
}
catch (AlreadyExistsException)
{
    scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
    var newStackName = PromptUserForStackName();
    return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
}
catch (Exception ex)
{
    scenarioLogger.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="stackId">The ID of the CloudFormation stack.</param>
/// <param name="cloudFormationClient">The CloudFormation client.</param>
/// <param name="scenarioLogger">The logger.</param>
/// <returns>True if the stack was created successfully.</returns>
private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
{

```

```
int retryCount = 0;
const int maxRetries = 30;
const int retryDelay = 10000;

while (retryCount < maxRetries)
{
    var describeStacksRequest = new DescribeStacksRequest
    {
        StackName = stackId
    };

    var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

    if (describeStacksResponse.Stacks.Count > 0)
    {
        if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
        {
            return true;
        }
        if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
        {
            return false;
        }
    }

    Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
    await Task.Delay(retryDelay);
    retryCount++;
}

scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
return false;
}

/// <summary>
/// Gets the outputs from the CloudFormation stack.
/// </summary>
```

```

    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>

```

```

    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
    IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
            This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
            cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
            deleted.");
                return true;
            }
            else
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
            '{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
            CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>

```

```
/// <param name="scenarioLogger">The logger.</param>
/// <returns>True if the stack was deleted successfully.</returns>
private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
{
    int retryCount = 0;
    const int maxRetries = 30;
    const int retryDelay = 10000;

    while (retryCount < maxRetries)
    {
        var describeStacksRequest = new DescribeStacksRequest
        {
            StackName = stackName
        };

        try
        {
            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count == 0 ||
                describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
            {
                return true;
            }
        }
        catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
        {
            return true;
        }

        Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
        await Task.Delay(retryDelay);
        retryCount++;
    }

    scenarioLogger.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>
    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>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrEmpty(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}

```

Una clase contenedora para los métodos AWS IoT del SDK.

```

/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
        ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)

```

```

        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
    if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }

```



```

    }
}

/// <summary>
/// Attaches a certificate to an IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing.</param>
/// <param name="certificateArn">The ARN of the certificate to attach.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
{
    try
    {
        var request = new AttachThingPrincipalRequest
        {
            ThingName = thingName,
            Principal = certificateArn
        };

        await _amazonIoT.AttachThingPrincipalAsync(request);
        _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Updates an IoT Thing with attributes.
/// </summary>
/// <param name="thingName">The name of the Thing to update.</param>
/// <param name="attributes">Dictionary of attributes to add.</param>

```

```

    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {

```

```

        EndpointType = "iot:Data-ATS"
    };

    var response = await _amazonIoT.DescribeEndpointAsync(request);
    _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
    return response.EndpointAddress;
}
catch (Amazon.IoT.Model.ThrottlingException ex)
{
    _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
    return null;
}
catch (Exception ex)
{
    _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
    return null;
}
}

/// <summary>
/// Lists all certificates associated with the account.
/// </summary>
/// <returns>List of certificate information, or empty list if listing failed.</
returns>
public async Task<List<Certificate>> ListCertificatesAsync()
{
    try
    {
        var request = new ListCertificatesRequest();
        var response = await _amazonIoT.ListCertificatesAsync(request);

        _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
        return response.Certificates;
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return new List<Certificate>();
    }
}

```

```

        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
}

```

```

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)

```

```

{
    try
    {
        var request = new CreateTopicRuleRequest
        {
            RuleName = ruleName,
            TopicRulePayload = new TopicRulePayload
            {
                Sql = "SELECT * FROM 'topic/subtopic'",
                Description = $"Rule created by .NET example: {ruleName}",
                Actions = new List<Amazon.IoT.Model.Action>
                {
                    new Amazon.IoT.Model.Action
                    {
                        Sns = new SnsAction
                        {
                            TargetArn = snsTopicArn,
                            RoleArn = roleArn
                        }
                    }
                },
                RuleDisabled = false
            }
        };

        await _amazonIoT.CreateTopicRuleAsync(request);
        _logger.LogInformation($"Created IoT rule {ruleName}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
    {
        _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Deletes an IoT topic rule.

```

```

    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)

```

```

        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");

```



```

        return await SetupIndexAndRetrySearchAsync(queryString);
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return new List<ThingDocument>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
        return new List<ThingDocument>();
    }
}

/// <summary>
/// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
/// </summary>
/// <param name="queryString">The search query string.</param>
/// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
{
    try
    {
        // Update indexing configuration to REGISTRY mode
        _logger.LogInformation("Setting up IoT search indexing
configuration...");
        await _amazonIoT.UpdateIndexingConfigurationAsync(
            new UpdateIndexingConfigurationRequest()
            {
                ThingIndexingConfiguration = new ThingIndexingConfiguration()
                {
                    ThingIndexingMode = ThingIndexingMode.REGISTRY
                }
            });

        _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

        // Wait for the index to be set up - this can take some time
        const int maxRetries = 10;
    }
}

```

```

        const int retryDelaySeconds = 10;

        for (int attempt = 1; attempt <= maxRetries; attempt++)
        {
            try
            {
                _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                // Try to get the current indexing configuration to see if it's
ready
                var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                {
                    // Try the search again
                    var request = new SearchIndexRequest
                    {
                        QueryString = queryString
                    };

                    var response = await _amazonIoT.SearchIndexAsync(request);
                    _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                    return response.Things;
                }
            }
            catch (Amazon.IoT.Model.IndexNotReadyException)
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");

                continue;
            }
            catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");

                continue;
            }
        }
    }

```

```

    }

    _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
    return new List<ThingDocument>();
}
catch (Exception ex)
{
    _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
    return new List<ThingDocument>();
}
}

/// <summary>
/// Detaches a certificate from an IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing.</param>
/// <param name="certificateArn">The ARN of the certificate to detach.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
{
    try
    {
        var request = new DetachThingPrincipalRequest
        {
            ThingName = thingName,
            Principal = certificateArn
        };

        await _amazonIoT.DetachThingPrincipalAsync(request);
        _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {

```

```

        _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
        return false;
    }
}

/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
    try
    {
        // First, update the certificate to inactive state
        var updateRequest = new UpdateCertificateRequest
        {
            CertificateId = certificateId,
            NewStatus = CertificateStatus.INACTIVE
        };
        await _amazonIoT.UpdateCertificateAsync(updateRequest);

        // Then delete the certificate
        var deleteRequest = new DeleteCertificateRequest
        {
            CertificateId = certificateId
        };

        await _amazonIoT.DeleteCertificateAsync(deleteRequest);
        _logger.LogInformation($"Deleted certificate {certificateId}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
        return false;
    }
}

```

```

    }
}

/// <summary>
/// Deletes an IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteThingAsync(string thingName)
{
    try
    {
        var request = new DeleteThingRequest
        {
            ThingName = thingName
        };

        await _amazonIoT.DeleteThingAsync(request);
        _logger.LogInformation($"Deleted Thing {thingName}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
        return false;
    }
}

/// <summary>
/// Lists IoT Things with pagination support.
/// </summary>
/// <returns>List of Things, or empty list if listing failed.</returns>
public async Task<List<ThingAttribute>> ListThingsAsync()
{
    try
    {
        // Use pages of 10.
        var request = new ListThingsRequest()

```

```

        {
            MaxResults = 10
        };
        var response = await _amazonIoT.ListThingsAsync(request);

        // Since there is not a built-in paginator, use the NextMarker to
paginate.
        bool hasMoreResults = true;

        var things = new List<ThingAttribute>();
        while (hasMoreResults)
        {
            things.AddRange(response.Things);

            // If NextMarker is not null, there are more results. Get the next
page of results.
            if (!String.IsNullOrEmpty(response.NextMarker))
            {
                request.Marker = response.NextMarker;
                response = await _amazonIoT.ListThingsAsync(request);
            }
            else
                hasMoreResults = false;
        }

        _logger.LogInformation($"Retrieved {things.Count} Things");
        return things;
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return new List<ThingAttribute>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
        return new List<ThingAttribute>();
    }
}
}

```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [AttachThingPrincipal](#)
 - [CreateKeysAndCertificate](#)
 - [CreateThing](#)
 - [CreateTopicRule](#)
 - [DeleteCertificate](#)
 - [DeleteThing](#)
 - [DeleteTopicRule](#)
 - [DescribeEndpoint](#)
 - [DescribeThing](#)
 - [DetachThingPrincipal](#)
 - [ListCertificates](#)
 - [ListThings](#)
 - [SearchIndex](#)
 - [UpdateIndexingConfiguration](#)
 - [UpdateThing](#)

Acciones

AttachThingPrincipal

En el siguiente ejemplo de código, se muestra cómo utilizar `AttachThingPrincipal`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
```

Acciones

```
/// Attaches a certificate to an IoT Thing.
```

653

```

    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [AttachThingPrincipal](#) la Referencia AWS SDK para .NET de la API.

CreateKeysAndCertificate

En el siguiente ejemplo de código, se muestra cómo utilizar CreateKeysAndCertificate.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Creates a device certificate for AWS IoT.
/// </summary>
/// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
{
    try
    {
        var request = new CreateKeysAndCertificateRequest
        {
            SetAsActive = true
        };

        var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
        _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
        return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
        return null;
    }
}
```

- Para obtener más información sobre la API, consulta [CreateKeysAndCertificate](#) la Referencia AWS SDK para .NET de la API.

CreateThing

En el siguiente ejemplo de código, se muestra cómo utilizar CreateThing.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Creates an AWS IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing to create.</param>
/// <returns>The ARN of the Thing created, or null if creation failed.</returns>
public async Task<string?> CreateThingAsync(string thingName)
{
    try
    {
        var request = new CreateThingRequest
        {
            ThingName = thingName
        };

        var response = await _amazonIoT.CreateThingAsync(request);
        _logger.LogInformation($"Created Thing {thingName} with ARN {response.ThingArn}");
        return response.ThingArn;
    }
    catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
    {
        _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
        return null;
    }
}
```

```

        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [CreateThing](#) la Referencia AWS SDK para .NET de la API.

CreateTopicRule

En el siguiente ejemplo de código, se muestra cómo utilizar CreateTopicRule.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Creates an IoT topic rule.
/// </summary>
/// <param name="ruleName">The name of the rule.</param>
/// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
/// <param name="roleArn">The ARN of the IAM role.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
{
    try
    {
        var request = new CreateTopicRuleRequest
        {
            RuleName = ruleName,
            TopicRulePayload = new TopicRulePayload
            {

```

```

        Sql = "SELECT * FROM 'topic/subtopic'",
        Description = $"Rule created by .NET example: {ruleName}",
        Actions = new List<Amazon.IoT.Model.Action>
        {
            new Amazon.IoT.Model.Action
            {
                Sns = new SnsAction
                {
                    TargetArn = snsTopicArn,
                    RoleArn = roleArn
                }
            },
            RuleDisabled = false
        }
    };

    await _amazonIoT.CreateTopicRuleAsync(request);
    _logger.LogInformation($"Created IoT rule {ruleName}");
    return true;
}
catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
{
    _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
    return false;
}
catch (Exception ex)
{
    _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
    return false;
}
}

```

- Para obtener más información sobre la API, consulta [CreateTopicRule](#) la Referencia AWS SDK para .NET de la API.

DeleteCertificate

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteCertificate.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
    try
    {
        // First, update the certificate to inactive state
        var updateRequest = new UpdateCertificateRequest
        {
            CertificateId = certificateId,
            NewStatus = CertificateStatus.INACTIVE
        };
        await _amazonIoT.UpdateCertificateAsync(updateRequest);

        // Then delete the certificate
        var deleteRequest = new DeleteCertificateRequest
        {
            CertificateId = certificateId
        };

        await _amazonIoT.DeleteCertificateAsync(deleteRequest);
        _logger.LogInformation($"Deleted certificate {certificateId}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot delete certificate - resource not found: {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
    }
}
```

```
    {  
        _logger.LogError($"Couldn't delete certificate. Here's why:  
{ex.Message}");  
        return false;  
    }  
}
```

- Para obtener más información sobre la API, consulta [DeleteCertificate](#) la Referencia AWS SDK para .NET de la API.

DeleteThing

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteThing.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// Deletes an IoT Thing.  
/// </summary>  
/// <param name="thingName">The name of the Thing to delete.</param>  
/// <returns>True if successful, false otherwise.</returns>  
public async Task<bool> DeleteThingAsync(string thingName)  
{  
    try  
    {  
        var request = new DeleteThingRequest  
        {  
            ThingName = thingName  
        };  
  
        await _amazonIoT.DeleteThingAsync(request);  
        _logger.LogInformation($"Deleted Thing {thingName}");  
        return true;  
    }  
}
```

```

        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
}

```

- Para obtener más información sobre la API, consulta [DeleteThing](#) la Referencia AWS SDK para .NET de la API.

DescribeEndpoint

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeEndpoint.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Gets the AWS IoT endpoint URL.
/// </summary>
/// <returns>The endpoint URL, or null if retrieval failed.</returns>
public async Task<string?> DescribeEndpointAsync()
{
    try
    {
        var request = new DescribeEndpointRequest
        {
            EndpointType = "iot:Data-ATS"
        };
    }
}

```

```

        var response = await _amazonIoT.DescribeEndpointAsync(request);
        _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
        return response.EndpointAddress;
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
        return null;
    }
}

```

- Para obtener más información sobre la API, consulta [DescribeEndpoint](#) la Referencia AWS SDK para .NET de la API.

DetachThingPrincipal

En el siguiente ejemplo de código, se muestra cómo utilizar `DetachThingPrincipal`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Detaches a certificate from an IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing.</param>
/// <param name="certificateArn">The ARN of the certificate to detach.</param>

```



```
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
{
    try
    {
        var request = new DetachThingPrincipalRequest
        {
            ThingName = thingName,
            Principal = certificateArn
        };

        await _amazonIoT.DetachThingPrincipalAsync(request);
        _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [DetachThingPrincipal](#) la Referencia AWS SDK para .NET de la API.

ListCertificates

En el siguiente ejemplo de código, se muestra cómo utilizar `ListCertificates`.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Lists all certificates associated with the account.
/// </summary>
/// <returns>List of certificate information, or empty list if listing failed.</
returns>
public async Task<List<Certificate>> ListCertificatesAsync()
{
    try
    {
        var request = new ListCertificatesRequest();
        var response = await _amazonIoT.ListCertificatesAsync(request);

        _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
        return response.Certificates;
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return new List<Certificate>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
        return new List<Certificate>();
    }
}
```

- Para obtener más información sobre la API, consulta [ListCertificates](#) la Referencia AWS SDK para .NET de la API.

ListThings

En el siguiente ejemplo de código, se muestra cómo utilizar ListThings.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Lists IoT Things with pagination support.
/// </summary>
/// <returns>List of Things, or empty list if listing failed.</returns>
public async Task<List<ThingAttribute>> ListThingsAsync()
{
    try
    {
        // Use pages of 10.
        var request = new ListThingsRequest()
        {
            MaxResults = 10
        };
        var response = await _amazonIoT.ListThingsAsync(request);

        // Since there is not a built-in paginator, use the NextMarker to
paginate.
        bool hasMoreResults = true;

        var things = new List<ThingAttribute>();
        while (hasMoreResults)
        {
            things.AddRange(response.Things);

            // If NextMarker is not null, there are more results. Get the next
page of results.
            if (!String.IsNullOrEmpty(response.NextMarker))
            {
                request.Marker = response.NextMarker;
                response = await _amazonIoT.ListThingsAsync(request);
            }
        }
    }
}
```

```

        else
            hasMoreResults = false;
        }

        _logger.LogInformation($"Retrieved {things.Count} Things");
        return things;
    }
    catch (Amazon.IoT.Model.ThrottlingException ex)
    {
        _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
        return new List<ThingAttribute>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
        return new List<ThingAttribute>();
    }
}

```

- Para obtener más información sobre la API, consulta [ListThings](#) la Referencia AWS SDK para .NET de la API.

SearchIndex

En el siguiente ejemplo de código, se muestra cómo utilizar SearchIndex.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Searches for IoT Things using the search index.
/// </summary>
/// <param name="queryString">The search query string.</param>

```

```

    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>

```

```

    /// Sets up the indexing configuration and retries the search after waiting for
    the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
    setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
    queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
                    _amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
                    == ThingIndexingMode.REGISTRY)

```

```

        {
            // Try the search again
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
            return response.Things;
        }
    }
    catch (Amazon.IoT.Model.IndexNotReadyException)
    {
        // Index still not ready, continue waiting
        _logger.LogInformation("Index still not ready, continuing to
wait...");
        continue;
    }
    catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
    {
        // Index still not ready, continue waiting
        _logger.LogInformation("Index still not ready, continuing to
wait...");
        continue;
    }
}

_logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
return new List<ThingDocument>();
}
catch (Exception ex)
{
    _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
    return new List<ThingDocument>();
}
}

```

- Para obtener más información sobre la API, consulta [SearchIndex](#) la Referencia AWS SDK para .NET de la API.

UpdateThing

En el siguiente ejemplo de código, se muestra cómo utilizar UpdateThing.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Updates an IoT Thing with attributes.
/// </summary>
/// <param name="thingName">The name of the Thing to update.</param>
/// <param name="attributes">Dictionary of attributes to add.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
{
    try
    {
        var request = new UpdateThingRequest
        {
            ThingName = thingName,
            AttributePayload = new AttributePayload
            {
                Attributes = attributes,
                Merge = true
            }
        };

        await _amazonIoT.UpdateThingAsync(request);
        _logger.LogInformation($"Updated Thing {thingName} with attributes");
        return true;
    }
    catch (Amazon.IoT.Model.ResourceNotFoundException ex)
    {
```



```
        _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [UpdateThing](#) la Referencia AWS SDK para .NET de la API.

AWS IoT data ejemplos que utilizan SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK para .NET (v4) con AWS IoT data.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Acciones](#)

Acciones

GetThingShadow

En el siguiente ejemplo de código, se muestra cómo utilizar GetThingShadow.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Gets the Thing's shadow information.
/// </summary>
/// <param name="thingName">The name of the Thing.</param>
/// <returns>The shadow data as a string, or null if retrieval failed.</returns>
public async Task<string?> GetThingShadowAsync(string thingName)
{
    try
    {
        var request = new GetThingShadowRequest
        {
            ThingName = thingName
        };

        var response = await _amazonIotData.GetThingShadowAsync(request);
        using var reader = new StreamReader(response.Payload);
        var shadowData = await reader.ReadToEndAsync();

        _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
        return shadowData;
    }
    catch (Amazon.IotData.Model.ResourceNotFoundException ex)
    {
        _logger.LogError($"Cannot get Thing shadow - resource not found: {ex.Message}");
        return null;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't get Thing shadow. Here's why: {ex.Message}");
        return null;
    }
}
```

- Para obtener más información sobre la API, consulta [GetThingShadow](#) la Referencia AWS SDK para .NET de la API.

UpdateThingShadow

En el siguiente ejemplo de código, se muestra cómo utilizar UpdateThingShadow.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Updates the Thing's shadow with new state information.
/// </summary>
/// <param name="thingName">The name of the Thing.</param>
/// <param name="shadowPayload">The shadow payload in JSON format.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
{
    try
    {
        var request = new UpdateThingShadowRequest
        {
            ThingName = thingName,
            Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
        };

        await _amazonIotData.UpdateThingShadowAsync(request);
        _logger.LogInformation($"Updated shadow for Thing {thingName}");
        return true;
    }
    catch (Amazon.IotData.Model.ResourceNotFoundException ex)
    {
```

```
        _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [UpdateThingShadow](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon Redshift con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon Redshift.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)

Introducción

Hello Amazon Redshift

En el siguiente ejemplo de código se muestra cómo empezar a utilizar Amazon Redshift.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Main method to run the Hello Amazon Redshift example.
/// </summary>
/// <param name="args">Command line arguments (not used).</param>
public static async Task Main(string[] args)
{
    var redshiftClient = new AmazonRedshiftClient();

    Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

    var clusters = new List<Cluster>();

    try
    {
        // Use pagination to retrieve all clusters.
        var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

        await foreach (var response in clustersPaginator.Responses)
        {
            if (response.Clusters != null)
                clusters.AddRange(response.Clusters);
        }

        Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

        foreach (var cluster in clusters)
        {
```

```
        Console.WriteLine($"{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
    }
}
catch (AmazonRedshiftException ex)
{
    Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred: {ex.Message}");
}
}
```

- Para obtener más información sobre la API, consulta [DescribeClusters](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crear un clúster de Redshift.
- Enumerar las bases de datos del clúster.
- Crear una tabla denominada "Movies".
- Rellenar la tabla "Movies".
- Consultar la tabla "Movies" por año.
- Modificar el clúster de Redshift.
- Eliminar el clúster de Amazon Redshift.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree una clase contenedora de Redshift para administrar las operaciones.

```

/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,

```

```

        NodeType = nodeType,
        NumberOfNodes = 1,
        ClusterType = "single-node"
    };

    var response = await _redshiftClient.CreateClusterAsync(request);
    Console.WriteLine($"Created cluster {clusterIdentifier}");
    return response.Cluster;
}
catch (ClusterAlreadyExistsException ex)
{
    Console.WriteLine($"Cluster already exists: {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
    throw;
}
}

/// <summary>
/// Describe Amazon Redshift clusters.
/// </summary>
/// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
/// <returns>A list of clusters.</returns>
public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
{
    try
    {
        var clusters = new List<Cluster>();
        var request = new DescribeClustersRequest();
        if (!string.IsNullOrEmpty(clusterIdentifier))
        {
            request.ClusterIdentifier = clusterIdentifier;
        }

        var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
        await foreach (var response in clustersPaginator.Responses)
        {
            if (response.Clusters != null)

```



```

        clusters.AddRange(response.Clusters);
    }

    Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
    foreach (var cluster in clusters)
    {
        Console.WriteLine($"{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
    }

    return clusters;
}
catch (ClusterNotFoundException ex)
{
    Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
    throw;
}
}

/// <summary>
/// Modify an Amazon Redshift cluster.
/// </summary>
/// <param name="clusterIdentifier">The identifier for the cluster.</param>
/// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
preferredMaintenanceWindow)
{
    try
    {
        var request = new ModifyClusterRequest
        {
            ClusterIdentifier = clusterIdentifier,
            PreferredMaintenanceWindow = preferredMaintenanceWindow
        };

```

```

        var response = await _redshiftClient.ModifyClusterAsync(request);
        Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
        return true;
    }
    catch (ClusterNotFoundException ex)
    {
        Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
        return false;
    }
}

/// <summary>
/// Delete an Amazon Redshift cluster without a final snapshot.
/// </summary>
/// <param name="clusterIdentifier">The identifier for the cluster.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
{
    try
    {
        var request = new DeleteClusterRequest
        {
            ClusterIdentifier = clusterIdentifier,
            SkipFinalClusterSnapshot = true
        };

        var response = await _redshiftClient.DeleteClusterAsync(request);
        Console.WriteLine($"The {clusterIdentifier} was deleted");
        return true;
    }
    catch (ClusterNotFoundException ex)
    {
        Console.WriteLine($"Cluster not found: {ex.Message}");
        return false;
    }
    catch (Exception ex)

```

```

        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");

```

```

        throw;
    }
}

/// <summary>
/// Create a table in the Redshift database.
/// </summary>
/// <param name="clusterIdentifier">The cluster identifier.</param>
/// <param name="database">The database name.</param>
/// <param name="dbUser">The database user.</param>
/// <returns>The statement ID.</returns>
public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
{
    try
    {
        var sqlStatement = @"
            CREATE TABLE Movies (
                id INTEGER PRIMARY KEY,
                title VARCHAR(250) NOT NULL,
                year INTEGER NOT NULL
            );

        var request = new ExecuteStatementRequest
        {
            ClusterIdentifier = clusterIdentifier,
            Database = database,
            DbUser = dbUser,
            Sql = sqlStatement
        };

        var response = await _redshiftDataClient.ExecuteStatementAsync(request);
        await WaitForStatementToCompleteAsync(response.Id);
        Console.WriteLine("Table created: Movies");
        return response.Id;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
    {
        Console.WriteLine($"Validation error: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
    }
}

```

```

        throw;
    }
}

/// <summary>
/// Insert a record into the Movies table using parameterized query.
/// </summary>
/// <param name="clusterIdentifier">The cluster identifier.</param>
/// <param name="database">The database name.</param>
/// <param name="dbUser">The database user.</param>
/// <param name="id">The movie ID.</param>
/// <param name="title">The movie title.</param>
/// <param name="year">The movie year.</param>
/// <returns>The statement ID.</returns>
public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
    int id, string title, int year)
{
    try
    {
        var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

        var request = new ExecuteStatementRequest
        {
            ClusterIdentifier = clusterIdentifier,
            Database = database,
            DbUser = dbUser,
            Sql = sqlStatement,
            Parameters = new List<SqlParameter>
            {
                new SqlParameter { Name = "id", Value = id.ToString() },
                new SqlParameter { Name = "title", Value = title },
                new SqlParameter { Name = "year", Value = year.ToString() }
            }
        };

        var response = await _redshiftDataClient.ExecuteStatementAsync(request);
        await WaitForStatementToCompleteAsync(response.Id);
        Console.WriteLine($"Inserted: {title} ({year})");
        return response.Id;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
    {

```

```

        Console.WriteLine($"Validation error: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Query movies by year using parameterized query.
/// </summary>
/// <param name="clusterIdentifier">The cluster identifier.</param>
/// <param name="database">The database name.</param>
/// <param name="dbUser">The database user.</param>
/// <param name="year">The year to query.</param>
/// <returns>A list of movie titles.</returns>
public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
    string dbUser, int year)
{
    try
    {
        var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

        var request = new ExecuteStatementRequest
        {
            ClusterIdentifier = clusterIdentifier,
            Database = database,
            DbUser = dbUser,
            Sql = sqlStatement,
            Parameters = new List<SqlParameter>
            {
                new SqlParameter { Name = "year", Value = year.ToString() }
            }
        };

        var response = await _redshiftDataClient.ExecuteStatementAsync(request);
        Console.WriteLine($"The identifier of the statement is {response.Id}");

        await WaitForStatementToCompleteAsync(response.Id);

        var results = await GetStatementResultAsync(response.Id);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Validation error: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
        throw;
    }
}

```

```

        var movieTitles = new List<string>();

        foreach (var row in results)
        {
            if (row.Count > 0)
            {
                var title = row[0].StringValue;
                Console.WriteLine($"The Movie title field is {title}");
                movieTitles.Add(title);
            }
        }

        return movieTitles;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
    {
        Console.WriteLine($"Validation error: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
        throw;
    }
}

/// <summary>
/// Describe a statement execution.
/// </summary>
/// <param name="statementId">The statement ID.</param>
/// <returns>The statement description.</returns>
public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
{
    try
    {
        var request = new DescribeStatementRequest
        {
            Id = statementId
        };

        var response = await
_redshiftDataClient.DescribeStatementAsync(request);
        return response;
    }
}

```

```

    }
    catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
    {
        Console.WriteLine($"Statement not found: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
        throw;
    }
}

/// <summary>
/// Get the results of a statement execution.
/// </summary>
/// <param name="statementId">The statement ID.</param>
/// <returns>A list of result rows.</returns>
public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
{
    try
    {
        var request = new GetStatementResultRequest
        {
            Id = statementId
        };

        var response = await
_redshiftDataClient.GetStatementResultAsync(request);
        return response.Records;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
    {
        Console.WriteLine($"Statement not found: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
        throw;
    }
}
}

```



```

    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);
    }

```

```

        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:\\:ss} - Waiting for
cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
{totalElapsed:mm\\:\\:ss}");
    }
}

```

Ejecute un escenario interactivo en el que se demuestren los fundamentos de Redshift.

```

/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
    }
}

```

```

        .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenariorAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenariorAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"=====");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"=====");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))

```

```

        userName = userInput;

Console.WriteLine("=====");
        Console.WriteLine("Please enter a user password for the cluster
(default is AwsUser1000):");
        var passwordInput = Console.ReadLine();
        if (!string.IsNullOrEmpty(passwordInput))
            userPassword = passwordInput;

Console.WriteLine("=====");

        // Step 2: Get cluster identifier
        Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
        var clusterInput = Console.ReadLine();
        if (!string.IsNullOrEmpty(clusterInput))
            clusterIdentifier = clusterInput;
    }
    else
    {
        Console.WriteLine($"Using default values: userName={userName},
clusterIdentifier={clusterIdentifier}");
    }

    // Step 3: Create Redshift cluster
    await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
userName, userPassword);

Console.WriteLine("=====");

    // Step 4: Wait for cluster to become available

Console.WriteLine("=====");
    await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

Console.WriteLine("=====");

    // Step 5: List databases

Console.WriteLine("=====");
    Console.WriteLine($" When you created {clusterIdentifier}, the dev
database is created by default and used in this scenario.");

```

```

        Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
        Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

Console.WriteLine("=====

Console.WriteLine("=====
        Console.WriteLine($"List databases in {clusterIdentifier}");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("=====

        // Step 6: Create Movies table

Console.WriteLine("=====
        Console.WriteLine("Now you will create a table named Movies.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("=====

        // Step 7: Populate the Movies table

Console.WriteLine("=====
        Console.WriteLine("Populate the Movies table using the Movies.json
file.");

```

```

        if (IsInteractive)
        {
            Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
            Console.WriteLine("Please enter a value between 50 and 200.");
            Console.Write("Enter a value: ");

            var recordCountInput = Console.ReadLine();
            if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
            {
                recordCount = inputCount;
            }
            else
            {
                Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
            }
        }
        else
        {
            Console.WriteLine($"Using default record count: {recordCount}");
        }

        await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
        Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("=====

        // Step 8 & 9: Query movies by year

Console.WriteLine("=====
        Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

        if (IsInteractive)
        {
            Console.Write("Enter a year: ");
            var yearInput = Console.ReadLine();
            if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)

```

```

        {
            year = inputYear;
        }
        else
        {
            Console.WriteLine($"Invalid input. Using default value of
{year}.");
        }
    }
    else
    {
        Console.WriteLine($"Using default year: {year}");
    }

    await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
userName, year);

    Console.WriteLine("=====

    // Step 10: Modify the cluster

    Console.WriteLine("=====
        Console.WriteLine("Now you will modify the Redshift cluster.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

    Console.WriteLine("=====

    // Step 11 & 12: Delete cluster confirmation

    Console.WriteLine("=====
        if (IsInteractive)
        {
            Console.WriteLine("Would you like to delete the Amazon Redshift
cluster? (y/n)");
            var deleteResponse = Console.ReadLine();
            if (deleteResponse?.ToLower() == "y")
            {

```

```

        await
Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
    }
}
else
{
    Console.WriteLine("Deleting the Amazon Redshift cluster...");
    await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
}

Console.WriteLine("=====

Console.WriteLine("=====
    Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=====
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
        Console.WriteLine("Deleting the Amazon Redshift cluster...");
        await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
        throw;
    }
}

/// <summary>
/// Populate the Movies table with data from the JSON file.
/// </summary>
/// <param name="clusterIdentifier">The cluster identifier.</param>
/// <param name="database">The database name.</param>
/// <param name="dbUser">The database user.</param>
/// <param name="recordCount">Number of records to insert.</param>
private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
{
    if (!File.Exists(_moviesFilePath))
    {
        throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
    }
}

```



```

        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}

```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [CreateCluster](#)
 - [DescribeClusters](#)
 - [DescribeStatement](#)
 - [ExecuteStatement](#)

- [GetStatementResult](#)
- [ListDatabasesPaginator](#)
- [ModifyCluster](#)

Acciones

CreateCluster

En el siguiente ejemplo de código, se muestra cómo utilizar CreateCluster.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Create a new Amazon Redshift cluster.
/// </summary>
/// <param name="clusterIdentifier">The identifier for the cluster.</param>
/// <param name="databaseName">The name of the database.</param>
/// <param name="masterUsername">The master username.</param>
/// <param name="masterUserPassword">The master user password.</param>
/// <param name="nodeType">The node type for the cluster.</param>
/// <returns>The cluster that was created.</returns>
public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
    string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
{
    try
    {
        var request = new CreateClusterRequest
        {
            ClusterIdentifier = clusterIdentifier,
            DBName = databaseName,
            MasterUsername = masterUsername,
            MasterUserPassword = masterUserPassword,
```

```

        NodeType = nodeType,
        NumberOfNodes = 1,
        ClusterType = "single-node"
    };

    var response = await _redshiftClient.CreateClusterAsync(request);
    Console.WriteLine($"Created cluster {clusterIdentifier}");
    return response.Cluster;
}
catch (ClusterAlreadyExistsException ex)
{
    Console.WriteLine($"Cluster already exists: {ex.Message}");
    throw;
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
    throw;
}
}

```

- Para obtener más información sobre la API, consulta [CreateCluster](#) la Referencia AWS SDK para .NET de la API.

DeleteCluster

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteCluster.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Delete an Amazon Redshift cluster without a final snapshot.
/// </summary>
/// <param name="clusterIdentifier">The identifier for the cluster.</param>

```

```
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
{
    try
    {
        var request = new DeleteClusterRequest
        {
            ClusterIdentifier = clusterIdentifier,
            SkipFinalClusterSnapshot = true
        };

        var response = await _redshiftClient.DeleteClusterAsync(request);
        Console.WriteLine($"The {clusterIdentifier} was deleted");
        return true;
    }
    catch (ClusterNotFoundException ex)
    {
        Console.WriteLine($"Cluster not found: {ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [DeleteCluster](#) la Referencia AWS SDK para .NET de la API.

DescribeClusters

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeClusters.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
    specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }

            return clusters;
        }
    }

```

```

    }
    catch (ClusterNotFoundException ex)
    {
        Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [DescribeClusters](#) la Referencia AWS SDK para .NET de la API.

DescribeStatement

En el siguiente ejemplo de código, se muestra cómo utilizar DescribeStatement.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Describe a statement execution.
/// </summary>
/// <param name="statementId">The statement ID.</param>
/// <returns>The statement description.</returns>
public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
{
    try
    {

```

```
        var request = new DescribeStatementRequest
        {
            Id = statementId
        };

        var response = await
_redshiftDataClient.DescribeStatementAsync(request);
        return response;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
    {
        Console.WriteLine($"Statement not found: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
        throw;
    }
}
```

- Para obtener más información sobre la API, consulta [DescribeStatement](#) la Referencia AWS SDK para .NET de la API.

GetStatementResult

En el siguiente ejemplo de código, se muestra cómo utilizar `GetStatementResult`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Get the results of a statement execution.
/// </summary>
```

```
/// <param name="statementId">The statement ID.</param>
/// <returns>A list of result rows.</returns>
public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
{
    try
    {
        var request = new GetStatementResultRequest
        {
            Id = statementId
        };

        var response = await
_redshiftDataClient.GetStatementResultAsync(request);
        return response.Records;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
    {
        Console.WriteLine($"Statement not found: {ex.Message}");
        throw;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
        throw;
    }
}
```

- Para obtener más información sobre la API, consulta [GetStatementResult](#) la Referencia AWS SDK para .NET de la API.

ListDatabases

En el siguiente ejemplo de código, se muestra cómo utilizar ListDatabases.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// List databases in a Redshift cluster.
/// </summary>
/// <param name="clusterIdentifier">The cluster identifier.</param>
/// <param name="dbUser">The database user.</param>
/// <param name="dbName">The database name for authentication.</param>
/// <returns>A list of database names.</returns>
public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string dbName)
{
    try
    {
        var request = new ListDatabasesRequest
        {
            ClusterIdentifier = clusterIdentifier,
            DbUser = dbUser,
            Database = dbName
        };

        var response = await _redshiftDataClient.ListDatabasesAsync(request);
        var databases = new List<string>();

        foreach (var database in response.Databases)
        {
            Console.WriteLine($"The database name is : {database}");
            databases.Add(database);
        }

        return databases;
    }
    catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
    {
        Console.WriteLine($"Validation error: {ex.Message}");
        throw;
    }
}
```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
        throw;
    }
}

```

- Para obtener más información sobre la API, consulta [ListDatabases](#) la Referencia AWS SDK para .NET de la API.

ModifyCluster

En el siguiente ejemplo de código, se muestra cómo utilizar `ModifyCluster`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Modify an Amazon Redshift cluster.
/// </summary>
/// <param name="clusterIdentifier">The identifier for the cluster.</param>
/// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
/// <returns>True if successful.</returns>
public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
preferredMaintenanceWindow)
{
    try
    {
        var request = new ModifyClusterRequest
        {
            ClusterIdentifier = clusterIdentifier,
            PreferredMaintenanceWindow = preferredMaintenanceWindow
        };
    }
}

```

```
        var response = await _redshiftClient.ModifyClusterAsync(request);
        Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
        return true;
    }
    catch (ClusterNotFoundException ex)
    {
        Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
        return false;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [ModifyCluster](#) la Referencia AWS SDK para .NET de la API.

Ejemplos de Amazon S3 con SDK para .NET (v4)

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el uso de la versión AWS SDK para .NET 4 con Amazon S3.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

En cada ejemplo se incluye un enlace al código de origen completo, con instrucciones de configuración y ejecución del código en el contexto.

Temas

- [Introducción](#)
- [Conceptos básicos](#)
- [Acciones](#)
- [Escenarios](#)

Introducción

Introducción a Amazon S3

El siguiente ejemplo de código muestra cómo empezar a utilizar Amazon S3.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Hello Amazon Simple Storage Service
/// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var s3Client = new AmazonS3Client();

        try
        {
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));

            // Use the built-in paginator to list buckets
```

```
var request = new ListBucketsRequest();
var paginator = s3Client.Paginators.ListBuckets(request);

var buckets = new List<S3Bucket>();

await foreach (var response in paginator.Responses)
{
    buckets.AddRange(response.Buckets);
}

if (buckets.Any())
{
    Console.WriteLine($"Found {buckets.Count} S3 buckets:");
    Console.WriteLine();

    foreach (var bucket in buckets)
    {
        Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
        Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-MM-dd HH:mm:ss UTC}");
        Console.WriteLine();
    }
}
else
{
    Console.WriteLine("No S3 buckets found in your account.");
}

Console.WriteLine("Hello S3 completed successfully.");
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"S3 service error occurred: {ex.Message}");
}
catch (Exception ex)
{
    Console.WriteLine($"Couldn't list S3 buckets. Here's why: {ex.Message}");
}
}
```

- Para obtener más información sobre la API, consulta [ListBuckets](#) la Referencia AWS SDK para .NET de la API.

Conceptos básicos

Conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Creación de un bucket y cargar un archivo en el bucket.
- Descargar un objeto desde un bucket.
- Copiar un objeto en una subcarpeta de un bucket.
- Obtención de una lista de los objetos de un bucket.
- Eliminación del bucket y todos los objetos que incluye.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en el que se muestran las características de Amazon S3.

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
```

```

        .AddTransient<S3Wrapper>()
        .AddLogging(builder => builder.AddConsole()))
    .Build();

_logger = LoggerFactory.Create(builder => builder.AddConsole())
    .CreateLogger<S3_Basics>();

_s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

var sepBar = new string('-', 45);

Console.WriteLine(sepBar);
Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
Console.WriteLine("procedures. This application will:");
Console.WriteLine("\n\t1. Create a bucket");
Console.WriteLine("\n\t2. Upload an object to the new bucket");
Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
Console.WriteLine("\n\t4. List the items in the new bucket");
Console.WriteLine("\n\t5. Delete all the items in the bucket");
Console.WriteLine("\n\t6. Delete the bucket");
Console.WriteLine(sepBar);

await RunScenario(_s3Wrapper, _logger);

Console.WriteLine(sepBar);
Console.WriteLine("The Amazon S3 scenario has successfully completed.");
Console.WriteLine(sepBar);
}

/// <summary>
/// Run the S3 Basics scenario with injected dependencies.
/// </summary>
/// <param name="s3Wrapper">The S3 wrapper instance.</param>
/// <param name="scenarioLogger">The logger instance.</param>
/// <returns>A Task object.</returns>
public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
{
    string bucketName = BucketName;
    string filePath = TempFilePath;
    string keyName = string.Empty;

    var sepBar = new string('-', 45);

```

```
try
{
    // Create a bucket.
    Console.WriteLine($"\\n{sepBar}");
    Console.WriteLine("\\nCreate a new Amazon S3 bucket.\\n");
    Console.WriteLine(sepBar);

    if (IsInteractive)
    {
        Console.Write("Please enter a name for the new bucket: ");
        bucketName = Console.ReadLine();
    }
    else
    {
        Console.WriteLine($"Using bucket name: {bucketName}");
    }

    var success = await s3Wrapper.CreateBucketAsync(bucketName);
    if (success)
    {
        Console.WriteLine($"Successfully created bucket: {bucketName}.\\n");
    }
    else
    {
        Console.WriteLine($"Could not create bucket: {bucketName}.\\n");
    }

    Console.WriteLine(sepBar);
    Console.WriteLine("Upload a file to the new bucket.");
    Console.WriteLine(sepBar);

    if (IsInteractive)
    {
        // Get the local path and filename for the file to upload.
        while (string.IsNullOrEmpty(filePath))
        {
            Console.Write("Please enter the path and filename of the file to
upload: ");

            filePath = Console.ReadLine();

            // Confirm that the file exists on the local computer.
            if (!File.Exists(filePath))
            {
```



```

        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
        filePath = string.Empty;
    }
}
else
{
    // Use the public variable if set, otherwise create a temp file
    if (!string.IsNullOrEmpty(TempFilePath))
    {
        filePath = TempFilePath;
        Console.WriteLine($"Using provided test file: {filePath}");
    }
    else
    {
        // Create a temporary test file for non-interactive mode
        filePath = Path.GetTempFileName();
        var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
        await File.WriteAllTextAsync(filePath, testContent);
        Console.WriteLine($"Created temporary test file: {filePath}");
    }
}

// Get the file name from the full path.
keyName = Path.GetFileName(filePath);

success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
filePath);

if (success)
{
    Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
}
else
{
    Console.WriteLine($"Could not upload {keyName}.\n");
}

// Set up download path
string downloadPath = string.Empty;

```

```
        if (IsInteractive)
        {
            // Now get a new location where we can save the file.
            while (string.IsNullOrEmpty(downloadPath))
            {
                // First get the path to which the file will be downloaded.
                Console.WriteLine("Please enter the path where the file will be
downloaded: ");
                downloadPath = Console.ReadLine();

                // Confirm that the file doesn't already exist on the local
computer.
                if (File.Exists($"{downloadPath}\\{keyName}"))
                {
                    Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                    downloadPath = string.Empty;
                }
            }
        }
        else
        {
            downloadPath = Path.GetTempPath();
            var downloadFile = Path.Combine(downloadPath, keyName);
            if (File.Exists(downloadFile))
            {
                File.Delete(downloadFile);
            }

            Console.WriteLine($"Using download path: {downloadPath}");
        }

        // Download an object from a bucket.
        success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);

        if (success)
        {
            Console.WriteLine($"Successfully downloaded {keyName}.\n");
        }
        else
        {
            Console.WriteLine($"Sorry, could not download {keyName}.\n");
        }
    }
}
```

```
// Copy the object to a different folder in the bucket.
string folderName = string.Empty;

if (IsInteractive)
{
    while (string.IsNullOrEmpty(folderName))
    {
        Console.Write("Please enter the name of the folder to copy your
object to: ");
        folderName = Console.ReadLine();
    }
}
else
{
    folderName = "test-folder";
    Console.WriteLine($"Using folder name: {folderName}");
}

await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
folderName);

// List the objects in the bucket.
await s3Wrapper.ListBucketContentsAsync(bucketName);

// Delete the contents of the bucket.
if (IsInteractive)
{
    Console.WriteLine("Press <Enter> when you are ready to delete the
bucket contents.");
    _ = Console.ReadLine();
}

var deleteContentsSuccess = await
s3Wrapper.DeleteBucketContentsAsync(bucketName);
if (deleteContentsSuccess)
{
    Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
}
else
{
    Console.WriteLine($"Sorry, could not delete contents of
{bucketName}.\n");
}
```

```
    }

    if (IsInteractive)
    {
        // Deleting the bucket too quickly after separately deleting its
contents can
        // cause an error that the bucket isn't empty. To delete contents
and bucket in one
        // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
        Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
        _ = Console.ReadLine();
    }
    else
    {
        // Add a small delay for non-interactive mode to ensure objects are
fully deleted.
        Console.WriteLine("Waiting a moment for objects to be fully
deleted...");
        await Task.Delay(2000);
    }

    // Delete the bucket.
    var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
    if (deleteSuccess)
    {
        Console.WriteLine($"Successfully deleted {bucketName}.\n");
    }
    else
    {
        Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
    }

    // Clean up temporary files in non-interactive mode
    if (!IsInteractive)
    {
        try
        {
            if (File.Exists(filePath))
            {
                File.Delete(filePath);
                Console.WriteLine("Cleaned up temporary test file.");
            }
        }
    }
}
```

```

        var downloadFile = Path.Combine(downloadPath, keyName);
        if (File.Exists(downloadFile))
        {
            File.Delete(downloadFile);
            Console.WriteLine("Cleaned up downloaded test file.");
        }
    }
    catch (Exception ex)
    {
        scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
    }
}
catch (Exception ex)
{
    scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

    // Clean up on error - delete bucket if it exists
    try
    {
        if (!string.IsNullOrEmpty(bucketName))
        {
            await s3Wrapper.DeleteBucketContentsAsync(bucketName);
            await s3Wrapper.DeleteBucketAsync(bucketName);
        }
    }
    catch (Exception cleanupEx)
    {
        scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
    }

    // Clean up temporary files in non-interactive mode
    if (!IsInteractive)
    {
        try
        {
            if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
            {
                File.Delete(filePath);
            }
        }
        catch (Exception fileCleanupEx)

```

```

        {
            scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
temporary files during error handling.");
        }
    }

    throw;
}
}
}

```

Una clase contenedora para métodos del SDK de Amazon S3.

```

using Amazon.S3;
using Amazon.S3.Model;

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
    }

    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of

```

```

/// the bucket creation process.</returns>
public async Task<bool> CreateBucketAsync(string bucketName)
{
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
        };

        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>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>
public async Task<bool> UploadFileAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    try
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,

```

```

        Key = objectName,
        FilePath = filePath,
    };

    var response = await _amazonS3.PutObjectAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
    return false;
}
}

/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    var request = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
    };

    using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

    try
    {
        // Save object to local file
    }
}

```



```

        await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error saving {objectName}: {ex.Message}");
        return false;
    }
}

```

```

/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
        {
            SourceBucket = bucketName,
            SourceKey = objectName,
            DestinationBucket = bucketName,
            DestinationKey = $"{folderName}\\{objectName}",
        };
        var response = await _amazonS3.CopyObjectAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error copying object: '{ex.Message}'");
    }
}

```

```

        return false;
    }
}

/// <summary>
/// Shows how to list the objects in an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket for which to list.
/// <param name="printList">True to print out the list.
/// <returns>The collection of objects.</returns>
public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
{
    try
    {
        var request = new ListObjectsV2Request
        {
            BucketName = bucketName,
            MaxKeys = 5,
        };

        if (printList)
        {
            Console.WriteLine("-----");
            Console.WriteLine($"Listing the contents of {bucketName}:");
            Console.WriteLine("-----");
        }

        var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
        {
            BucketName = bucketName,
        });
        var s3objects = new List<S3Object>();
        await foreach (var response in listObjectsV2Paginator.Responses)
        {
            if (response.S3Objects != null)
            {
                s3objects.AddRange(response.S3Objects);
            }
        }
    }
}

```

```

        if (printList)
        {
            Console.WriteLine($"Number of Objects: {s3Objects.Count}");
            foreach (var entry in s3Objects)
            {
                Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
            }
        }

        return s3Objects;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error encountered on server. Message: '{ex.Message}'
getting list of objects.");
        return null;
    }
}

/// <summary>
/// Delete all of the objects stored in an existing Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket from which the
/// contents will be deleted.</param>
/// <returns>A boolean value that represents the success or failure of
/// deleting all of the objects in the bucket.</returns>
public async Task<bool> DeleteBucketContentsAsync(string bucketName)
{
    // Iterate over the contents of the bucket and delete all objects.
    try
    {
        // Delete all objects in the bucket.
        var deleteList = await ListBucketContentsAsync(bucketName, false);
        if (deleteList != null && deleteList.Any())
        {
            await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
            });
        }
    }
}

```

```

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting objects: {ex.Message}");
        return false;
    }
}

/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
}

```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
 - [CopyObject](#)
 - [CreateBucket](#)

- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

Acciones

CopyObject

En el siguiente ejemplo de código, se muestra cómo utilizar CopyObject.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
```

```

    {
        SourceBucket = bucketName,
        SourceKey = objectName,
        DestinationBucket = bucketName,
        DestinationKey = $"{folderName}\\{objectName}",
    };
    var response = await _amazonS3.CopyObjectAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error copying object: '{ex.Message}'");
    return false;
}
}

```

- Para obtener más información sobre la API, consulta [CopyObject](#) la Referencia AWS SDK para .NET de la API.

CreateBucket

En el siguiente ejemplo de código, se muestra cómo utilizar CreateBucket.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Shows how to create a new Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <returns>A boolean value representing the success or failure of
/// the bucket creation process.</returns>
public async Task<bool> CreateBucketAsync(string bucketName)

```

```
{
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
        };

        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 obtener más información sobre la API, consulta [CreateBucket](#) la Referencia AWS SDK para .NET de la API.

CreatePresignedPost

En el siguiente ejemplo de código, se muestra cómo utilizar CreatePresignedPost.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crear una URL POST previamente firmada.

```
/// <summary>
/// Create a presigned POST URL with conditions.
/// </summary>
/// <param name="s3Client">The Amazon S3 client.</param>
```

```

    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
            }
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }

```


- Para obtener más información sobre la API, consulta [CreatePresignedPost](#) la Referencia AWS SDK para .NET de la API.

DeleteBucket

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteBucket.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [DeleteBucket](#) la Referencia AWS SDK para .NET de la API.

DeleteObjects

En el siguiente ejemplo de código, se muestra cómo utilizar DeleteObjects.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Delete all of the objects stored in an existing Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket from which the
/// contents will be deleted.</param>
/// <returns>A boolean value that represents the success or failure of
/// deleting all of the objects in the bucket.</returns>
public async Task<bool> DeleteBucketContentsAsync(string bucketName)
{
    // Iterate over the contents of the bucket and delete all objects.
    try
    {
        // Delete all objects in the bucket.
        var deleteList = await ListBucketContentsAsync(bucketName, false);
        if (deleteList != null && deleteList.Any())
        {
            await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
            });
        }

        return true;
    }
}
```

```
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- Para obtener más información sobre la API, consulta [DeleteObjects](#) la Referencia AWS SDK para .NET de la API.

GetObject

En el siguiente ejemplo de código, se muestra cómo utilizar `GetObject`.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
{
```

```
var request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = objectName,
};

using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

try
{
    // Save object to local file
    await response.WriteResponseStreamToFileAsync($"{filePath}\{objectName}", true, CancellationToken.None);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
catch (AmazonS3Exception ex)
{
    Console.WriteLine($"Error saving {objectName}: {ex.Message}");
    return false;
}
```

- Para obtener más información sobre la API, consulta [GetObject](#) la Referencia AWS SDK para .NET de la API.

ListObjectsV2

En el siguiente ejemplo de código, se muestra cómo utilizar ListObjectsV2.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
```

```

    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("-----");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("-----");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3objects.Count}");
                foreach (var entry in s3objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }
        }
    }

```

```

        }

        return s3objects;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error encountered on server. Message: '{ex.Message}'
getting list of objects.");
        return null;
    }
}

```

- Para obtener más información sobre la API, consulta la [ListObjectsversión 2](#) en la referencia de la AWS SDK para .NET API.

PutObject

En el siguiente ejemplo de código, se muestra cómo utilizar PutObject.

SDK para .NET (v4)

Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/// <summary>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>

```

```
public async Task<bool> UploadFileAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    try
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
        };

        var response = await _amazonS3.PutObjectAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
        return false;
    }
}
```

- Para obtener más información sobre la API, consulta [PutObject](#) la Referencia AWS SDK para .NET de la API.

Escenarios

Crear una URL prefirmada

En el siguiente ejemplo de código se muestra cómo crear una URL prefirmada para Amazon S3 y cargar un objeto.

SDK para .NET (v4)

 Note

Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Crea y usa un POST prefirmado URLs para subirlo directamente al navegador.

```
/// <summary>
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetupServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }
}
```



```

/// <summary>
/// Perform the actions defined for the Amazon S3 Presigned POST scenario.
/// </summary>
/// <param name="args">Command line arguments.</param>
/// <returns>A Task object.</returns>
public static async Task Main(string[] args)
{
    _isInteractive = !args.Contains("--non-interactive");

    // Set up dependency injection for Amazon S3
    using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
        services.AddAWSService<IAmazonS3>()
            .AddTransient<S3Wrapper>()
            .AddHttpClient()
        )
    .Build();

    SetUpServices(host);

    try
    {
        // Display overview
        _uiMethods.DisplayOverview();
        _uiMethods.PressEnter(_isInteractive);

        // Step 1: Create bucket
        await CreateBucketAsync();
        _uiMethods.PressEnter(_isInteractive);

        // Step 2: Create presigned URL
        _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
        var response = await CreatePresignedPostAsync();
        _uiMethods.PressEnter(_isInteractive);

        // Step 3: Display URL and fields
        _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
        DisplayPresignedPostFields(response);
        _uiMethods.PressEnter(_isInteractive);

        // Step 4: Upload file
    }
}

```

```

        _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
URL");

        await UploadFileAsync(response);
        _uiMethods.PressEnter(_isInteractive);

        // Step 5: Verify file exists
        await VerifyFileExistsAsync();
        _uiMethods.PressEnter(_isInteractive);

        // Step 6: Cleanup
        _uiMethods.DisplayTitle("Step 6: Clean up resources");
        await CleanupAsync();

        _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
successfully!");
        _uiMethods.PressEnter(_isInteractive);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error in scenario");
        Console.WriteLine($"Error: {ex.Message}");

        // Attempt cleanup if there was an error
        if (!string.IsNullOrEmpty(_bucketName))
        {
            _uiMethods.DisplayTitle("Cleaning up resources after error");
            await _s3Wrapper.DeleteBucketAsync(_bucketName);
            Console.WriteLine($"Cleaned up bucket: {_bucketName}");
        }
    }
}

/// <summary>
/// Create an S3 bucket for the scenario.
/// </summary>
private static async Task CreateBucketAsync()
{
    _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

    // Generate a default bucket name for the scenario
    var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

    // Prompt user for bucket name or use default in non-interactive mode

```

```

        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}')": "",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrEmpty(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");

        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }

    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
    private static async Task<CreatePresignedPostResponse>
    CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);

        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.

```

```

    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");

        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
{uploadResult.StatusCode}");
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
{uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {

```

```

        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);

            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>

```

```

private static async Task VerifyFileExistsAsync()
{
    _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

    Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

    try
    {
        var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
        _objectKey!);

        Console.WriteLine($"File verification successful! File exists in the
bucket.");
        Console.WriteLine($"File size: {metadata.ContentLength} bytes");
        Console.WriteLine($"File type: {metadata.Headers.ContentType}");
        Console.WriteLine($"Last modified: {metadata.LastModified}");
    }
    catch (AmazonS3Exception ex) when (ex.StatusCode ==
System.Net.HttpStatusCode.NotFound)
    {
        Console.WriteLine($"Error: File was not found in the bucket.");
        throw;
    }
}

private static void DisplayPresignedPostFields(CreatePresignedPostResponse
response)
{
    Console.WriteLine($"Presigned POST URL: {response.Url}");
    Console.WriteLine("Form fields to include:");

    foreach (var field in response.Fields)
    {
        Console.WriteLine($"  {field.Key}: {field.Value}");
    }
}

/// <summary>
/// Clean up resources created by the scenario.
/// </summary>
private static async Task CleanupAsync()
{
    if (!string.IsNullOrEmpty(_bucketName))

```

```
{  
    Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");  
    bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);  
  
    if (result)  
    {  
        Console.WriteLine("Bucket deleted successfully");  
    }  
    else  
    {  
        Console.WriteLine("Failed to delete bucket - it may have been  
already deleted");  
    }  
}  
}
```

Migración de su proyecto para el AWS SDK para .NET

En esta sección se proporciona información sobre las tareas de migración para la AWS SDK para .NET versión 4 del sistema e instrucciones sobre cómo realizarlas. Para obtener información sobre la migración a versiones anteriores (por ejemplo, la migración de la V2 a la V3 o de una subversión de la V3 a otra), [consulte Migración del proyecto](#) en la [Guía para AWS SDK para .NET desarrolladores](#) (versión 3).

Consulte también [Novedades](#) y [Plataformas admitidas](#).

Temas

- [Migración a la versión 4 del AWS SDK para .NET](#)
- [Migración desde .NET Standard 1.3](#)

Migración a la versión 4 del AWS SDK para .NET

La AWS SDK para .NET versión 4 (V4) presenta un número significativo de cambios importantes con respecto a la versión 3 (V3) del SDK. En este tema se describen los cambios más importantes de la versión 4 y los posibles trabajos que podría ser necesario realizar para migrar el entorno o el código desde la V3. Para obtener información adicional sobre otros cambios importantes en el SDK, consulta los siguientes recursos:

- El número del rastreador de desarrollo en GitHub: <https://github.com/aws/aws-sdk-net/issues/3362>
- La entrada del blog [Vista previa 1 de la AWS SDK para .NET V4](#).
- La entrada del blog [Vista previa 4 de la AWS SDK para .NET V4](#).
- La entrada del blog [Disponibilidad general de la AWS SDK para .NET V4.0](#).

.NET Framework

El objetivo de .NET Framework 3.5 se ha eliminado de la versión 4 del AWS SDK para .NET. Como resultado, el SDK ya no es compatible con .NET Framework 3.5. Esta versión del SDK está compilada con .NET Framework 4.7.2 y se ejecuta en el entorno de ejecución .NET 4.0. Para obtener más información, consulte [Plataformas compatibles](#).

Tipos de valor

Las propiedades que utilizan tipos de valores en las clases que se utilizan para realizar solicitudes y respuestas se han modificado para utilizar tipos de valores que admiten valores nulos. Se han modificado las propiedades con los siguientes tipos:

- `boolse` ha cambiado a `bool`?
- `doublese` ha cambiado a `double`?
- `intse` ha cambiado a `int`?
- `floatse` ha cambiado a `float`?
- `longse` ha cambiado a `long`?
- `Datetimese` ha cambiado a `Datetime`?

Para obtener información adicional sobre este cambio, consulte la entrada del blog [Vista previa 1 de la AWS SDK para .NET versión 4](#).

Colecciones

Las propiedades que utilizan colecciones en las clases que se utilizan para realizar solicitudes y respuestas ahora están predeterminadas como `null`. Como resultado, el código debe comprobar que una colección no es nula antes de intentar usarla. Por ejemplo:

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

El comportamiento de la V3 al inicializar las colecciones se puede restaurar `Amazon.AWSConfigs.InitializeCollections` configurando en `true`. Esta propiedad también existe en la versión 3 para los usuarios que desean probar este cambio de comportamiento antes de actualizar a la versión 4.

Para obtener más información sobre este cambio, consulte la entrada del blog [Vista previa 1 de la versión AWS SDK para .NET 4](#).

AWS Security Token Service (STS)

- El punto final regional

Cuando se utilizan proveedores de credenciales que dependen de ellos AWS STS, las llamadas siempre utilizan el punto final regional. Esto difiere de la versión 3 del SDK, que utilizaba la us-east-1 región de forma predeterminada cuando se ejecutaba en la partición pública, independientemente de la región configurada.

- La enumeración **StsRegionalEndpointsValue**

La `StsRegionalEndpointsValue` enumeración se eliminó del espacio de nombres [Amazon.Runtime](#). Se debe eliminar cualquier código que utilice esa enumeración.

- La **STSAssumeRoleAWSCredentials** clase

[El obsoleto STS Assume rol de proveedor de credenciales STSAssumeRoleAWSCredentials,, se ha eliminado de Amazon. SecurityToken](#) espacio de nombres. En su lugar, [AssumeRoleAWSCredentials](#) utilícelo desde [Amazon.Runtime](#).

Cambios relacionados con **ClientConfig**

El [Amazon.Runtime. ClientConfig](#) es la clase base de las clases de configuración de clientes de servicio, como [AmazonS3Config](#). Se realizaron los siguientes cambios en esta clase base.

- Modo de reintento predeterminado

El valor predeterminado `RetryMode` de la propiedad es `Standard` en lugar de. `Legacy` Como resultado, el `Legacy` valor se ha eliminado de [Amazon.Runtime. RequestRetryMode](#) enumeración.

- Modo de configuración predeterminado

El valor predeterminado de la `DefaultConfigurationMode` propiedad es `Standard` en lugar de `Legacy`. Como resultado, el `Legacy` valor se ha eliminado de [Amazon.Runtime. DefaultConfigurationMode](#) enumeración.

- ¿**ReadWriteTimeout** La propiedad?

La `ReadWriteTimeout` propiedad obsoleta se eliminó de todos los destinos excepto de .NET Framework 4.7.2.

Las AWSSDK.Extensions.NETCoreEl paquete .Setup NuGet

Las [AWSSDK.Extensions.NETCore NuGet El paquete .Setup](#) se ha actualizado para solucionar los problemas que estaban presentes en la versión 3 del SDK y para que el paquete sea seguro para el AOT nativo. Estos cambios se resumen a continuación. Para obtener información detallada, consulte el [PR 3353](#) en el [aws-sdk-net](#) repositorio de GitHub

- La **DefaultClientConfig** clase

La `DefaultClientConfig` clase ya no se hereda de la clase base de configuración del cliente de servicio [Amazon.Runtime.ClientConfig](#). Las propiedades relevantes de `ClientConfig` han replicado `DefaultClientConfig` utilizando tipos de valores que aceptan valores nulos. Este cambio nos permite detectar cuándo se ha establecido un valor al copiar los valores a la configuración que `DefaultClientConfig` se está creando para el cliente del servicio.

Un resultado concreto de este cambio es que ya no `DefaultClientConfig.HttpClientFactory` está disponible en la versión 4. En su lugar, use `AWSConfigs.HttpClientFactory`. Para obtener información adicional, consulte el [GitHub número 3790](#).

Otro resultado de este cambio es que se han eliminado las sobrecargas genéricas del método de `IConfiguration.GetAWSOptions` extensión que aceptaba un objeto de configuración de servicio. En su lugar, se debe utilizar la sobrecarga no genérica y el SDK se encargará automáticamente de rellenar los ajustes específicos del servicio. [Para obtener información adicional, consulta el número 3866. GitHub](#)

- AOT nativo

Se ha agregado al paquete un nuevo mecanismo para crear clientes de servicio que utiliza los métodos de interfaz estática de C# 11. Este cambio elimina la necesidad de realizar cargas de tipo ensamblado para crear instancias de clientes de servicio, incluida la manipulación de cadenas del nombre de la interfaz de servicio para calcular el tipo de cliente de servicio, lo que no es compatible con el AOT nativo. Este cambio solo está disponible para .NET 8 y versiones posteriores; las versiones anteriores siguen utilizando el mecanismo original.

Para obtener información adicional sobre este paquete en esta guía, consulte [AWSSDK.Extensions.NETCore.Configuración y IConfiguration](#). El código fuente de este paquete está en GitHub en <https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup>.

CookieSigner y UrlSigner

Las UrlSigner extensiones CookieSigner y de Amazon se CloudFront han trasladado a un paquete de extensiones independiente denominado [AWSSDK.Extensions.CloudFront.Firmantes](#). Este cambio es para admitir OpenSSL 3 y pasar a depender de .Cryptography. BouncyCastle

El código fuente de este paquete está en. GitHub <https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers>

DateTime frente a UTC DateTime

Algunas clases de la versión 3 tienen una DateTime propiedad marcada como «obsoleta» u «obsoleta», así como una propiedad UTC DateTime alternativa. En estas clases, se ha eliminado la DateTime propiedad obsoleta y se ha cambiado el nombre de la DateTime propiedad UTC por el nombre original de la DateTime propiedad.

A continuación se muestran algunos ejemplos de clases en las que se ha implementado este cambio.

- [DescribeSpotPriceHistoryRequest](#):
 - Se ha eliminado la StartTime propiedad obsoleta y se ha cambiado su nombre a "StartTime".
StartTimeUtc
 - Se ha eliminado la EndTime propiedad obsoleta y se ha cambiado su nombre a "EndTime".
EndTimeUtc
- [CreateFleetRequest](#)
 - Se ha eliminado la ValidFrom propiedad obsoleta y se ha cambiado su nombre a "ValidFrom".
ValidFromUtc
 - Se ha eliminado la ValidUntil propiedad obsoleta y se ha cambiado su nombre a "ValidUntil".
ValidUntilUtc

Este cambio puede provocar tiempos de compensación si una aplicación utiliza la DateTime propiedad original obsoleta. Se producirá un error en el momento de la compilación en el código que utilice la DateTime propiedad UTC.

DateTime análisis

La `DateTimeUnmarshaller` clase se ha actualizado. Esta clase ha estado analizando y devolviendo `DateTime` cadenas en hora local. En algunos casos, estos valores se volvían a convertir a UTC debido a una actualización previa, pero no siempre. Ahora, `DateTime` las cadenas que no están ordenadas se consideran UTC y se especificarán y desordenarán como UTC. Esta actualización incluye los siguientes cambios de comportamiento.

Algunas propiedades de la marca de tiempo que se basan en la `DateTime` clase se estaban analizando para convertirlas en horas locales. Estas incluían desagrupadores de respuestas para las marcas de tiempo y listas de marcas de tiempo para los formatos `TimestampFormat.ISO8601` `TimestampFormat.RFC822` `DateTime`. El análisis se ha actualizado para devolver las horas UTC en su lugar.

`ConvertFromUnixEpochSeconds` y `ConvertFromUnixEpochMilliseconds`

[ConvertFromUnixEpochMilliseconds](#) Los métodos [ConvertFromUnixEpochSeconds](#) y, que convierten los segundos de época de Unix en una `DateTime` estructura, devolvían la hora de Unix como hora local en lugar de como hora UTC. Estos métodos ahora devuelven la hora UTC.

Registro

La forma en que se habilita el inicio de sesión en el SDK se ha actualizado para la versión 4. El inicio de sesión en la consola y los diagnósticos del sistema funcionan igual que en la versión 3, es decir, se establece la `LoggingConfig.LogTo` propiedad de la [AWSConfigs](#) clase en `LoggingOptions.Console` o `LoggingOptions.SystemDiagnostics`. Se `log4net` ha eliminado la [LoggingOptions](#) opción `for`, junto con la lógica interna del SDK, que permitía utilizar la reflexión para adjuntarla a una instancia en memoria de `log4net`.

Para incluir el inicio de sesión del SDK en un marco de registro, se utiliza un paquete de adaptadores independiente para conectar el SDK con el marco de registro. [Usa el paquete `AWSSDK.Extensions.Logging.Log4` y el paquete `NetAdaptor.Extensions.Logging.log4net` `AWSSDK ILoggerMicrosoft.Extensions.LoggingPaquete de adaptadores para`](#). Los siguientes ejemplos de código muestran cómo configurar el registro en estos dos casos.

Ejemplo de configuración para log4net

Agregue el `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet paquete y llame al `ConfigureAWSSDKLogging` método estático desde `Log4NetAWSExtensions`

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Ejemplo de configuración para Microsoft.Extensions.Logging

Agregue el `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet paquete y llame al método de `ConfigureAWSSDKLogging` extensión desde la interfaz `ILoggerFactory`

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support para HTTP 2

Se ha añadido soporte para HTTP 2 para permitir la transmisión bidireccional. Para obtener más información, consulte [Support para HTTP 2](#).

Inicio de sesión único

El valor predeterminado de la `SupportsGettingNewToken` propiedad de la clase [SSOAWSCredentialsOptions](#) se ha cambiado de `true` a `false`. Si tiene aplicaciones que utilizan la [SSOAWSCredentials](#) clase para obtener credenciales de SSO, es posible que deba establecer la `Options.SupportsGettingNewToken` propiedad en `true`. Para ver ejemplos de esta configuración, consulte los [ejemplos de código](#) en [Tutorial de SSO utilizando únicamente aplicaciones .NET](#). Para obtener información adicional, consulte [PR 3737](#) en el [aws-sdk-net](#) GitHub repositorio.

Cambios específicos de DynamoDB

Los siguientes cambios son específicos de Amazon DynamoDB. Muchos de ellos son cambios importantes.

Para obtener información adicional sobre los cambios en DynamoDB en la versión 4 de AWS SDK para .NET la, consulte la entrada del [blog Vista previa](#) 4 de la versión 4. AWS SDK para .NET

Abrir para ver los elementos

Los cambios en la versión 4 del SDK para DynamoDB abordan algunos problemas relacionados con la capacidad de prueba, pero se centran principalmente en las bibliotecas de alto nivel:

- El [modelo de documentos de .NET](#), cuyo nombre aparece [DocumentModel](#) en el código.
- El [modelo de persistencia de objetos de .NET](#), cuyo nombre [DataModel](#) aparece en el código.

Para obtener información detallada sobre estos modos de programación, consulte [DynamoDB](#) esta guía.

Modelo de documento: excepción actualizada para una interfaz simulada **IAmazonDynamoDB**

En el modelo de documento anterior a la versión 4 del SDK, si se inicializaba una [tabla](#) con una interfaz simulada de [IAmazonDynamoDB](#), regresaba. `NullReferenceException` En su lugar, vuelve la versión 4 del SDK. `InvalidOperationException` TableLos métodos asíncronos deberían funcionar con un cliente simulado, pero es posible que aún veas excepciones al llamar a métodos síncronos desde ellos. .NET/Core/Standard

[Para obtener más información sobre este cambio, consulte PR 3388 en.](#) GitHub

Modelo de documento: **FromJson** y métodos **ToJson**

ToJsonLos métodos `FromJson` y de la clase [Document](#) ahora se utilizan `System.Text.Json` en lugar de `LitJson` para la serialización, y se `LitJson` han eliminado de la versión 4 del SDK. Una ventaja de su uso `System.Text.Json` es que este analizador admite el uso del `Decimal` tipo.NET, que permite una mayor precisión en las propiedades numéricas de punto flotante.

Modelo de persistencia de objetos: la clase **DynamoDBOperationConfig**

En el modelo de persistencia de objetos, se han realizado los siguientes cambios en la clase compartida de [Dynamo DBOperation Config](#):

- La clase se ha dividido en nuevas clases específicas para cada operación [SaveConfig](#), como, y [LoadConfig](#). [QueryConfig](#) Los métodos utilizados se DynamoDBOperationConfig han marcado como obsoletos y están sujetos a su eliminación en el futuro.

Para obtener más información sobre este cambio, consulte [PR 3421](#) en GitHub.

- Las DisableFetchingTableMetadata propiedades MetadataCachingMode y se han eliminado de la clase. Estas propiedades no se incluyeron en las nuevas clases específicas de operaciones mencionadas anteriormente. [Las propiedades eliminadas son valores de nivel de tabla que deben especificarse en la Context propiedad global de la clase DynamoDB o en la AWSConfigsclass Dynamo Config. DBContext](#)

[Para obtener más información sobre este cambio, consulte PR 3422 en.](#) GitHub

- La clase ya no hereda de la clase [Dynamo DBContext Config](#). Esto impide pasar un DynamoDBOperationConfig objeto al constructor de [Dynamo DBContext](#), donde algunas propiedades de la configuración específica de la operación (por ejemplo) no se OverrideTableName aplican.

Para obtener más información sobre este cambio, consulte [PR 3422](#) en. GitHub

Modelo de persistencia de objetos: polimorfismo

La DBPolymorphic TypeAttribute clase [Dynamo](#) se agregó al modelo de persistencia de objetos. Esta clase admite la serialización y deserialización de tipos polimórficos. [Para obtener más información, consulte PR 3643 en.](#) GitHub

Modelo de documento y modelo de persistencia de objetos: operaciones simulables

Se han agregado nuevas interfaces específicas de operación que permiten a los clientes simular las operaciones de DynamoDB. Los métodos de fábrica de la [IDynamoDBContext](#) interfaz se han actualizado para mostrar las nuevas interfaces.

Para obtener más información sobre este cambio, consulte [PR 3450](#) en GitHub.

- Modelo de persistencia de objetos
 - BatchGetSimule operaciones a través de las IMultiTableBatchGet interfaces IBatchGet y.
 - BatchWriteOperaciones simuladas a través de las IMultiTableBatchWrite interfaces IBatchWrite y.

- TransactGetOperaciones simuladas a través de las IMultiTableTransactGet interfaces ITransactGet y.
- TransactWriteOperaciones simuladas a través de las IMultiTableTransactWrite interfaces ITransactWrite y.
- Simulación Scan y Query operaciones a través de la IAsyncSearch interfaz.
- Modelo de documento
 - Simula Table operaciones a través de la ITable interfaz.
 - Simulación Scan y Query operaciones a través de la ISearch interfaz.
 - TransactWriteSimule operaciones a través de las IMultiTableDocumentTransactWrite interfaces IDocumentTransactWrite y.
 - TransactGetOperaciones simuladas a través de las IMultiTableDocumentTransactGet interfaces IDocumentTransactGet y.
 - BatchWriteOperaciones simuladas a través de las IMultiTableDocumentBatchWrite interfaces IDocumentBatchWrite y.
 - BatchGetOperaciones simuladas a través de las IMultiTableDocumentBatchGet interfaces IDocumentBatchGet y.

Modelo de documentos y modelo de persistencia de objetos: Support para AOT nativo

Una limitación del AOT nativo es la compatibilidad con tipos.NET anidados. En algunos casos, estos tipos anidados pueden pasar desapercibidos para el componente de recorte del compilador.NET. En este caso, es posible que reciba una excepción como: "» System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.

Puedes evitar esta limitación añadiendo un punto de la DynamicDependency ruta del código que informe al recortador de la dependencia del subtipo. El constructor del tipo.NET de nivel superior que se va a guardar es un lugar probable. El siguiente ejemplo de código muestra cómo utilizar el DynamicDependency atributo:

```
[DynamoDBTable("TestTable")]
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {
```

```

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
    public string Name { get; set; }

    public SubType SubType { get; set; }
}

class SubType
{
    public string SubName { get; set; }
}

```

Dinamo DBStreams

[Dynamo](#) se DBStreams ha eliminado del paquete [AWSSDK.DynamoDB](#) y está disponible en su propio NuGet paquete, [AWSSDK.Dynamo](#), y tiene su propio espacio de nombres [DBStreams](#), `Amazon.DynamoDBStreams`

Permitir la eliminación **TableNamePrefix** del valor

Ahora puede eliminar el valor de la `TableNamePrefix` propiedad en la clase [Dynamo DBContext Config](#) en el nivel de operación individual. Para obtener más información sobre este cambio, consulte [PR 3476](#) en GitHub

Propiedad **RetrieveDateTimeInUtc**

Para la clase [Dynamo DBContext Config](#), el valor predeterminado de la `RetrieveDateTimeInUtc` propiedad se ha cambiado a `true`.

Propiedad **DynamoDBContextTableNamePrefix**

Se ha eliminado la `DynamoDBContextTableNamePrefix` propiedad de la [AWSConfigs](#) clase [DynamoDB](#). Los usuarios deberían llamar `AWSConfigsDynamoDB.Context.TableNamePrefix` en lugar de

Cambios específicos de EC2

Los siguientes cambios son específicos de Amazon EC2. La mayoría o todos ellos son cambios importantes.

[Abrir para ver los artículos](#)

GetDecryptedPassword

[La GetDecryptedPassword extensión de Amazon EC2 se ha trasladado a un paquete de extensiones independiente denominado AWSSDK.Extensions.EC2.DecryptPassword. Este cambio es para admitir OpenSSL 3 y pasar a depender de .Cryptography. BouncyCastle](#)

El código fuente de este paquete está en. GitHub <https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword>

Support para Amazon EC2 IMDSv1

Se ha eliminado la compatibilidad con la versión 1 (IMDSv1) del servicio de metadatos de instancias. La versión 4 del SDK siempre usa la versión 2 (IMDSv2) del Servicio de Metadatos de Instancia al obtener las credenciales y otros metadatos del IMDS. Para obtener más información sobre el IMDS, consulte [Uso del IMDS en la](#) Guía del usuario de [Amazon EC2](#).

Elementos de programación que se han modificado o eliminado

- Se ha eliminado todo el `Amazon.EC2.Import` espacio de nombres y el código.
- Se ha eliminado todo el `Amazon.EC2.Util` espacio de nombres y el código, que incluyen las utilidades de AMI que se utilizaban para buscar EC2 AMIs para Windows.
- La `IpRanges` propiedad obsoleta se ha eliminado de la clase. [IpPermission](#) Utilice las `Ipv6Ranges` propiedades `Ipv4Ranges` o en su lugar.
- Se han eliminado de la [EC2InstanceMetadata](#) clase los siguientes campos obsoletos: `EC2_METADATA_SVC`, `EC2_METADATA_ROOT`, `EC2_USERDATA_ROOT`, `EC2_DYNAMICDATA_ROOT`, y `EC2_API_TOKEN_URL`.

Cambios específicos de S3

Los siguientes cambios son específicos de Amazon S3. La mayoría o todos ellos son cambios importantes.

Abrir para ver los artículos

Región de AWS us-east-1

Los clientes del servicio Amazon S3 configurados para la us-east-1 región ya no pueden acceder a los buckets de otras regiones. Se debe acceder a los buckets con los clientes de servicio de S3 configurados para la región en la que se encuentra el bucket.

Para obtener información adicional sobre este cambio, consulte la entrada del blog [Preview 4 of AWS SDK para .NET V4](#).

Cliente de cifrado S3

El cliente de cifrado Amazon S3, que se define en el espacio de `Amazon.S3.Encryption` nombres, se ha eliminado del [AWSSDKpaquete.S3](#). Este cliente se ha trasladado a su propio paquete denominado [Amazon.Extensions.S3.Encryption](#) y su documentación se encuentra en <https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html>. Para obtener información sobre cómo migrar, consulte [the section called “Migración del cliente de cifrado S3 \(V1 a V2\)”](#). Para obtener más información sobre el cifrado S3, consulte [Algoritmos de cifrado compatibles](#) en la [Guía para desarrolladores de Amazon S3 Encryption Client](#).

Directiva de etiquetado de S3 para **CopyObject**

La `TaggingDirective` propiedad ha sido expuesta como una propiedad pública de la [CopyObjectRequest](#) clase, que es utilizada por `AmazonS3Client.CopyObject` los métodos. Esta propiedad corresponde al `x-amz-tagging-directive` parámetro Amazon S3, tal como se define en la [CopyObject](#) acción.

La directiva de etiquetado ya no se establece automáticamente en COPY. Si un desarrollador no especifica una directiva de etiquetado, el backend de S3 asume automáticamente que es COPY, pero si un desarrollador establece explícitamente la propiedad en null, el valor no se establece en absoluto.

La **UseArnRegion** propiedad de la configuración de S3

La `UseArnRegion` propiedad de la clase [Amazon.S3.AmazonS3Config](#) se ha actualizado para que la variable de `AWS_S3_USE_ARN_REGION` entorno tenga prioridad sobre la configuración del archivo compartido. `s3_use_arn_region` AWS config [Para obtener más información sobre estas variables y configuraciones, consulte la referencia sobre la configuración en la Guía de referencia de herramientas y herramientas.AWS SDKs](#)

Barras diagonales iniciales para los métodos **CopyObject** y **CopyPart**

Las barras diagonales iniciales ya no se recortarán para Amazon S3 CopyObject y CopyPart sus métodos. La DisableTrimmingLeadingSlash propiedad se ha eliminado de las clases [CopyObjectRequest](#) y [CopyPartRequest](#).

Los **DoesS3BucketExist...** métodos

[Los DoesS3BucketExistAsync métodos obsoletos DoesS3BucketExist se han eliminado de la clase AmazonS3Util, que implementa la interfaz AmazonS3. ICore](#) Estos métodos se eliminaron porque siempre utilizan HTTP. En su lugar, utilice [doESS3 BucketExist V2](#) y [BucketExistdoESS3 V2Async](#).

El SDK siempre usa SigV4

La versión 4 del AWS SDK para .NET siempre usa la versión 4 de AWS Signature (SiGv4) para firmar las solicitudes. Este cambio produce los siguientes cambios relacionados:

- Se ha eliminado la UseSignatureVersion4 propiedad de la clase [AWSConfigsS3](#).
- SignatureVersionPropiedad de [Amazon.Runtime. ClientConfig](#) se ha eliminado la clase. Esta propiedad solo la utilizaba Amazon S3 por motivos de compatibilidad con versiones anteriores.
- Se ha eliminado la RegionEndpoint.Endpoint clase. Esto incluye la SignatureVersionOverride propiedad, que se utilizó para anular las versiones firmadas de Amazon S3. En su lugar, utilice el método específico del servicio `client.DetermineServiceOperationEndPoint()`.
- [Métodos actualizados: AmazonS3Util. PostUploadPostUploadSignedPolicy y S3. GetSignedPolicy](#) para usar SiGV4. Como consecuencia, se eliminó el `S3PostUploadSignedPolicy.GetSignedPolicyV4` método porque `GetSignedPolicy` ahora realiza la misma función. Además, se `GetSignedPolicy` ha asignado un tercer parámetro para el punto final de la región.

Los **PutACL** métodos **GetACL** y

Los PutACL métodos GetACL y de la clase [AmazonS3Client](#) se han marcado como obsoletos. Para acceder a la funcionalidad de estos métodos, utilice en su lugar los siguientes métodos nuevos: `GetBucketACL`, `PutBucketACL`, `GetObjectACL` y `PutObjectACL`

Se han eliminado los elementos de programación obsoletos

Se eliminaron varios elementos de programación de la implementación de Amazon S3 de la versión 4 del SDK, incluidos los valores de enumeración, los tipos, los métodos, los espacios de nombres, etc. Estos se enumeran a continuación, si no se trataron anteriormente, junto con las posibles medidas que puede tomar para proceder a su eliminación.

- La `DisableMD5Stream` propiedad ha sido retirada de la [TransferUtilityUploadRequest](#) clase. Utilice la propiedad `DisableDefaultChecksumValidation` en su lugar.

Además, la `CalculateContentMD5Header` propiedad ha sido eliminada de la `TransferUtilityUploadRequest` clase. Esta propiedad ya no es necesaria porque el SDK calcula una suma de comprobación de forma predeterminada.

- Las `ServerSideEncryptionKeyManagementServiceKeyId` propiedades `ServerSideEncryptionMethod` y se han eliminado de la [CopyPartRequest](#) clase. En su lugar, utilice las propiedades con los mismos nombres en la [InitiateMultipartUploadRequest](#) clase, que se utilizan en algunos de los `InitiateMultipartUpload...` métodos de la clase [AmazonS3Client](#).
- La `Expires` propiedad se ha eliminado de la clase. [GetObjectResponse](#) Utilice la propiedad `ExpiresString` en su lugar. Es posible que la cadena no tenga un formato de marca de tiempo válido, por lo que el código debería usar el `TryParse` método al convertirla a `DateTime`.
- [Se han eliminado Región de AWS los identificadores obsoletos de la enumeración de S3Region.](#)
- La `Prefix` propiedad se ha eliminado de la clase. [LifecycleRule](#) Utilice la propiedad `Filter` en su lugar.

Además, las `Transition` propiedades `NoncurrentVersionTransition` y propiedades se han eliminado de la `LifecycleRule` clase. En su lugar, utilice `Transitions` las colecciones `NoncurrentVersionTransitions` y.

- La `Event` propiedad se ha eliminado de la [TopicConfiguration](#) clase. En su lugar, usa la `Events` colección.
- `CalculateContentMD5` propiedad de encabezado. Ya no es necesario establecer esta propiedad porque el SDK calculará una suma de comprobación de forma predeterminada.
- La `Bucket` propiedad se ha eliminado de la [SelectObjectContentRequest](#) clase. Utilice la propiedad `BucketName` en su lugar.
- La `NumberOfUploadThreads` propiedad ha sido eliminada de la [TransferUtilityConfig](#) clase. En su lugar, utilice la `ConcurrentServiceRequests` propiedad.

Elementos de programación que se eliminaron

Se eliminaron varios elementos de programación de la versión 4 del SDK, incluidos los valores de enumeración, los tipos, los métodos, los espacios de nombres, etc. Estos se enumeran a continuación, si no se trataron anteriormente, junto con las posibles medidas que puede tomar para proceder a su eliminación.

Abrir para ver los artículos

El espacio de nombres **Amazon.Auth.AccessControlPolicy.ActionIdentifiers**

Se ha eliminado el `Amazon.Auth.AccessControlPolicy.ActionIdentifiers` espacio de nombres. Esto incluye los identificadores de acciones de IAM, que se definieron en la clase. `IdentityandAccessManagementActionIdentifiers` El código que usa estos identificadores de acción debe cambiarse para usar valores de cadena del nombre de la acción.

Para obtener información adicional, consulte [the section called “Creación de políticas administradas a partir de JSON”](#) la [descripción general de las políticas de JSON](#) en la Guía del [usuario de IAM](#).

La clase **ClientConfig**

El [Amazon.Runtime](#). `ClientConfig` es la clase base de las clases de configuración de clientes de servicio, como [AmazonS3Config](#). Se han eliminado los siguientes elementos de programación de esta clase.

- `DetermineDnsSuffix` Se han eliminado los métodos `DetermineServiceURL` y. En su lugar, utilice el `DetermineServiceOperationEndpoint` método del cliente de servicio; por ejemplo, [AmazonS3Client.DetermineServiceOperationEndpoint](#).
- La `ReadEntireResponse` propiedad ha sido retirada. En su lugar, utilice una de las siguientes opciones:
 - La `LogResponses` propiedad de [AWSConfigs.LoggingConfig](#) clase.
 - La `LogResponse` propiedad de la configuración del cliente; por ejemplo, [AmazonS3Config](#).

El espacio de nombres **Amazon.Runtime**

El espacio de nombres [Amazon.Runtime](#) se actualizó de la siguiente manera:

- La `ECSTaskCredentials` clase obsoleta se ha eliminado del espacio de nombres. En su lugar, utilice el [GenericContainerCredentials](#) proveedor, que también es compatible con [Amazon EKS Pod Identities](#).

- Las clases obsoletas `StoredProfileAWSCredentials` y `StoredProfileCredentials` las clases se han eliminado del espacio de nombres. [Utilice el Net SDKCredentials File o la SharedCredentialsFile](#) clase de `Amazon.Runtime.CredentialManagement` espacio de nombres en su lugar.
- El `HasCachedAccessTokenAvailable` método obsoleto de la [SSOAWSCredentials](#) clase se ha eliminado del espacio de nombres.
- La `EnvironmentAWSCredentials` clase obsoleta se ha eliminado del espacio de nombres. En su lugar, utilice la clase [AppConfigAWSCredentials](#).
- La `StoredProfileFederatedCredentials` clase obsoleta se ha eliminado del espacio de nombres. En su lugar, utilice la [clase federada. AWSCredentials](#)
- Se han eliminado del espacio de nombres las siguientes clases obsoletas: `EnvironmentVariableAWSEndpointDiscoveryEnabled`, `yProfileAWSEndpointDiscoveryEnabled`. `FallbackEndpointDiscoveryEnabledFactory`
- La `UseSigV4` propiedad obsoleta se ha eliminado de la [AmazonWebServiceRequest](#) clase. Utilice la propiedad `SignatureVersion` en su lugar.
- La `ProfileIniFile` clase del espacio de `Amazon.Runtime.Internal.Util` nombres tiene un método sobrecargado llamado `TryGetSection` Las versiones del método que no admiten el `out` parámetro `for se nestedProperties` han eliminado de la clase.
- Se ha eliminado la `EventBridgeSigner` clase obsoleta `Amazon.Runtime.Internal.Auth` del espacio de nombres.
- La propiedad de `Parameters` diccionario obsoleta se ha eliminado de la [WebServiceRequestEventArgs](#) clase. Utilice la propiedad `ParameteCollection` en su lugar.

BouncyCastle

La copia fuente de se BouncyCastle ha eliminado de la versión 4 del SDK.

La clase **StoredProfileSAMLCredentials**

La `StoredProfileSAMLCredentials` clase obsoleta de [Amazon. SecurityToken](#) Se ha eliminado el espacio de nombres. [SAML. En su lugar, utilice la AWSCredentials clase Federated en el espacio de nombres Amazon.Runtime.](#)

La clase **AWSSDKUtils**

Se han eliminado los siguientes métodos de la [AWSSDKUtils](#) clase: y. `ResolveResourcePath`
`ProtectEncodedSlash` `UrlEncode` `ConvertToUnixEpochMilliseconds`

La clase **ProfileManager**

La `ProfileManager` clase obsoleta se ha eliminado del espacio de nombres [Amazon.Util](#).
[Utilice el SDKCredentials archivo Net o SharedCredentialsFile la clase de Amazon.Runtime.CredentialManagement](#) espacio de nombres en su lugar.

La clase **AWSConfigs**

Se han eliminado de la [AWSConfigs](#) clase las siguientes propiedades obsoletas:
`LoggingResponseLogging`, y. `LogMetrics` Utilice la propiedad `LoggingConfig` en su lugar.

La clase **ConditionFactory**

Se ha eliminado de la [ConditionFactory](#) clase el método con la siguiente
firma: `NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Utilice el
[NewConditionUtc](#) método en su lugar.

CloudFront Utilidades de Amazon

Se han eliminado el espacio de `Amazon.CloudFront.Util` nombres y la
`AmazonCloudFrontUtil` clase obsoletos.

AWS IoT

En la [ListPrincipalThingsResponse](#) clase, se ha eliminado una personalización antigua que preveía
una `NextToken` anulación en favor de la paginación.

AWS Lambda

Se han eliminado los siguientes `Invoke . . .` métodos de la [AmazonLambdaClient](#) clase porque los
nombres eran confusos.

- Se ha eliminado el método V3 con la siguiente firma: `InvokeAsyncResponse`
`InvokeAsync(InvokeAsyncRequest)`. Este es un método sincrónico en la versión 3 del SDK.
En su lugar, utilícelo `InvokeResponse` `Invoke(InvokeRequest)` (para el procesamiento
sincrónico) o `Task InvokeAsync(InvokeRequest, CancellationToken)` (para el
procesamiento asíncrono).

- Se ha eliminado el método V3 con la siguiente firma: `Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)` Este es un método asíncrono en la versión 3 del SDK. En su lugar, use `Task InvokeAsync(InvokeRequest, CancellationToken)`.

Amazon SageMaker Runtime

Se han eliminado los constructores obsoletos [PayloadPart](#) de la clase.

Migración desde .NET Standard 1.3

El 27 de junio de 2019 Microsoft [terminó el soporte](#) para las versiones de .NET Core 1.0 y .NET Core 1.1. Tras este anuncio, el 31 de diciembre de 2020 AWS finalizó AWS SDK para .NET el soporte para .NET Standard 1.3.

AWS continuó proporcionando actualizaciones de servicio y correcciones de seguridad para el .NET Standard 1.3, que AWS SDK para .NET estaba destinado a la versión 1.3, hasta el 1 de octubre de 2020. Después de esa fecha, el objetivo de .NET Standard 1.3 pasó al modo de mantenimiento, lo que significaba que no se publicaban nuevas actualizaciones; solo AWS se aplicaban correcciones de errores críticos y parches de seguridad.

El 31 de diciembre de 2020, el soporte para .NET Standard 1.3 AWS SDK para .NET llegó a su fin. Después de esta fecha, no se han aplicado correcciones de errores o revisiones de seguridad. Los artefactos creados con ese objetivo siguen estando disponibles para su descarga NuGet.

Qué necesita

- Si está ejecutando aplicaciones que utilizan .NET Framework, no se verá afectado.
- Si está ejecutando aplicaciones que utilizan .NET Core 2.0 o superior, no se verá afectado.
- Si ejecuta aplicaciones con .NET Core 1.0 o .NET Core 1.1, migre las aplicaciones a una versión más reciente de .NET Core siguiendo las [instrucciones de migración de Microsoft](#). Recomendamos como mínimo la versión .NET Core 3.1.
- Si está ejecutando aplicaciones críticas para la empresa que no se pueden actualizar en este momento, puede continuar utilizando la versión actual de AWS SDK para .NET.

Si tiene preguntas o dudas, [póngase en contacto con AWS Support](#).

Seguridad para este AWS producto o servicio

La seguridad en la nube de Amazon Web Services (AWS) es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes. La seguridad es una responsabilidad compartida entre usted AWS y usted. En el [modelo de responsabilidad compartida](#), se habla de “seguridad de la nube” y “seguridad en la nube”:

Seguridad de la nube: AWS se encarga de proteger la infraestructura en la que se ejecutan todos los servicios que se ofrecen en la AWS nube y de proporcionarle servicios que pueda utilizar de forma segura. Nuestra responsabilidad en materia de seguridad es nuestra máxima prioridad AWS, y auditores externos comprueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los [programas de AWS conformidad](#).

Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice y otros factores, como la confidencialidad de sus datos, los requisitos de su organización y las leyes y reglamentos aplicables.

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Temas

- [Protección de datos en este AWS producto o servicio](#)
- [Gestión de identidad y acceso](#)
- [Validación de la conformidad de este AWS producto o servicio](#)
- [Resiliencia de este AWS producto o servicio](#)
- [Seguridad de la infraestructura para este AWS producto o servicio](#)
- [Imponer una versión mínima de TLS en el AWS SDK para .NET](#)
- [Migración del cliente de cifrado S3 \(de la V1 a la V2\)](#)
- [Migración de un cliente de cifrado S3 \(de V2 a V4\)](#)

Protección de datos en este AWS producto o servicio

El [modelo de](#) se aplica a protección de datos de este AWS producto o servicio. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan todos los Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las [Preguntas frecuentes sobre la privacidad de datos](#). Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el [Modelo de responsabilidad compartida de AWS y GDPR](#) en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte [Cómo trabajar con CloudTrail senderos](#) en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger la información confidencial almacenada en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte [Estándar de procesamiento de la información federal \(FIPS\) 140-3](#).

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja con este AWS producto o servicio u otro Servicios de AWS mediante la consola, la API o. AWS CLI AWS SDKs Cualquier dato que introduzca

en etiquetas o campos de formato libre utilizados para los nombres se pueden emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Gestión de identidad y acceso

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Temas

- [Público](#)
- [Autenticación con identidades](#)
- [Administración del acceso con políticas](#)
- [¿Cómo Servicios de AWS trabajar con IAM](#)
- [Solución de problemas de AWS identidad y acceso](#)

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice. AWS

Usuario del servicio: si Servicios de AWS solía hacer su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una función de AWS, consulte [Solución de problemas de AWS identidad y acceso](#) o consulte la guía del usuario de la Servicio de AWS que está utilizando.

Administrador de servicios: si está a cargo de AWS los recursos de su empresa, probablemente tenga acceso total a ellos AWS. Su trabajo consiste en determinar a qué AWS funciones y recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página

para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar la IAM AWS, consulte la guía del usuario del Servicio de AWS que está utilizando.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS. Para ver ejemplos de políticas AWS basadas en la identidad que puede utilizar en IAM, consulte la guía del usuario de la Servicio de AWS que está utilizando.

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe autenticarse como usuario de Usuario raíz de la cuenta de AWS IAM o asumir una función de IAM.

Puede iniciar sesión como una identidad federada con las credenciales de una fuente de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales. Google/Facebook Para obtener más información sobre el inicio de sesión, consulte [Cómo iniciar sesión en la Cuenta de AWS](#) en la Guía del usuario de AWS Sign-In .

Para el acceso programático, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte [AWS Signature Version 4 para solicitudes de API](#) en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear un Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario Cuenta de AWS raíz que tiene acceso completo a todos Servicios de AWS los recursos. Se recomienda encarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver las tareas que requieren credenciales de usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio empresarial, del proveedor de identidades web o al Directory Service que se accede Servicios de AWS mediante credenciales de una fuente de identidad. Las identidades federadas asumen roles que proporcionan credenciales temporales.

Para una administración de acceso centralizada, se recomienda AWS IAM Identity Center. Para obtener más información, consulte [¿Qué es el Centro de identidades de IAM?](#) en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de larga duración. Para obtener más información, consulte [Exigir a los usuarios humanos que utilicen la federación con un proveedor de identidad para acceder AWS mediante credenciales temporales](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. Para obtener más información, consulte [Casos de uso para usuarios de IAM](#) en la Guía del usuario de IAM.

Roles de IAM

Un [Rol de IAM](#) es una identidad con permisos específicos que proporciona credenciales temporales. Puede asumir un rol [cambiando de un rol de usuario a uno de IAM \(consola\)](#) o llamando a una AWS CLI operación de AWS API. Para obtener más información, consulte [Métodos para asumir un rol](#) en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuario federado, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Administración del acceso con políticas

AWS Para controlar el acceso, puede crear políticas y adjuntarlas a AWS identidades o recursos. Una política define los permisos cuando están asociados a una identidad o un recurso. AWS evalúa estas políticas cuando un director hace una solicitud. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte [Información general de políticas de JSON](#) en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte [Definición de permisos de IAM personalizados con políticas administradas por el cliente](#) en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte [Selección entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Los ejemplos incluyen las Políticas de confianza de roles de IAM y las Políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Debe [especificar una entidad principal](#) en una política basada en recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso () ACLs

Las listas de control de acceso (ACLs) controlan qué responsables (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios compatibles. AWS WAF ACLs Para obtener más información ACLs, consulte la [descripción general de la lista de control de acceso \(ACL\)](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer los permisos máximos otorgados por los tipos de políticas más comunes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.
- Políticas de control de servicios (SCPs): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte [Políticas de control de servicios](#) en la Guía del usuario de AWS Organizations .
- Políticas de control de recursos (RCPs): establece los permisos máximos disponibles para los recursos de tus cuentas. Para obtener más información, consulte [Políticas de control de recursos \(RCPs\)](#) en la Guía del AWS Organizations usuario.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener más información, consulte [Políticas de sesión](#) en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la [lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

¿Cómo Servicios de AWS trabajar con IAM

Para obtener una visión general de cómo Servicios de AWS trabajar con la mayoría de las funciones de IAM, consulte [AWS los servicios que funcionan con IAM en la Guía del usuario de IAM](#).

Para obtener información sobre cómo utilizar una función específica Servicio de AWS con IAM, consulte la sección de seguridad de la guía del usuario del servicio correspondiente.

Solución de problemas de AWS identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas habituales que pueden surgir al trabajar con un AWS IAM.

Temas

- [No estoy autorizado a realizar ninguna acción en AWS](#)
- [No estoy autorizado a realizar tareas como: PassRole](#)
- [Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos](#)

No estoy autorizado a realizar ninguna acción en AWS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM mateojackson intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *my-example-widget*, pero no tiene los permisos ficticios awes : *GetWidget*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
awes: GetWidget on resource: my-example-widget
```

En este caso, la política del usuario mateojackson debe actualizarse para permitir el acceso al recurso *my-example-widget* mediante la acción awes : *GetWidget*.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar tareas como: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción iam:PassRole, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS.

Algunos Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir la función al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado marymajor intenta utilizar la consola para realizar una acción en AWS. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS recursos

Se puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Se puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que respaldan las políticas basadas en recursos o las listas de control de acceso (ACLs), puedes usar esas políticas para permitir que las personas accedan a tus recursos.

Para obtener más información, consulte lo siguiente:

- Para saber si AWS es compatible con estas funciones, consulte. [¿Cómo Servicios de AWS trabajar con IAM](#)
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte [Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en](#) la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo [proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Validación de la conformidad de este AWS producto o servicio

Para saber si un programa de cumplimiento Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte [Servicios de AWS Alcance por programa](#) de de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de [AWS cumplimiento > Programas AWS](#) .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#).

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. Para obtener más información sobre su responsabilidad de conformidad al utilizarlos Servicios de AWS, consulte [AWS la documentación de seguridad](#).

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Resiliencia de este AWS producto o servicio

La infraestructura AWS global se basa en Regiones de AWS zonas de disponibilidad.

Regiones de AWS proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia.

Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

[Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS](#)

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Seguridad de la infraestructura para este AWS producto o servicio

Este AWS producto o servicio utiliza servicios gestionados y, por lo tanto, está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura, consulte [Seguridad AWS en la nube](#). Para diseñar su AWS

entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte [Protección de infraestructuras en un marco](#) de buena AWS arquitectura basado en el pilar de la seguridad.

Utiliza las llamadas a la API AWS publicadas para acceder a este AWS producto o servicio a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puedes utilizar [AWS Security Token Service](#) (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Este AWS producto o servicio sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) a los que da soporte. Para obtener información sobre la seguridad de los AWS servicios, consulte la [página de documentación sobre la seguridad del AWS servicio](#) y [AWS los servicios que se encuentran dentro del ámbito de aplicación de AWS las medidas de conformidad establecidas por el programa de conformidad](#).

Imponer una versión mínima de TLS en el AWS SDK para .NET

Para aumentar la seguridad al comunicarse con AWS los servicios, debe configurarlos AWS SDK para .NET para que utilicen TLS 1.2 o una versión posterior.

AWS SDK para .NET Utiliza el entorno de ejecución de .NET subyacente para determinar qué protocolo de seguridad utilizar. De forma predeterminada, las versiones actuales de .NET utilizan el último protocolo configurado que admite el sistema operativo. Su aplicación puede anular este comportamiento del SDK, pero no se recomienda hacerlo.

.NET Core

De forma predeterminada, .NET Core utiliza el último protocolo configurado que admite el sistema operativo. El AWS SDK para .NET no proporciona un mecanismo para anular esto.

Si está utilizando una versión de .NET Core anterior a la 2.1, le recomendamos encarecidamente que actualice la versión de .NET Core.

Consulte lo siguiente para obtener información específica de cada sistema operativo.

Windows

Las distribuciones modernas de Windows tienen la compatibilidad con TLS 1.2 [habilitada de forma predeterminada](#). Si utiliza Windows 7 SP1 o Windows Server 2008 R2 SP1, debe asegurarse de que la compatibilidad con TLS 1.2 esté habilitada en el registro, tal y como se describe en <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12>. Si está ejecutando una distribución anterior, debe actualizar el sistema operativo. Para obtener información sobre la compatibilidad con TLS 1.3 en Windows, eche un vistazo a la documentación de Microsoft más reciente para conocer las versiones de cliente o de servidor mínimas requeridas.

macOS

Si ejecuta .NET Core 2.1 o una versión posterior, TLS 1.2 está habilitado de forma predeterminada. TLS 1.2 es compatible con [OS X Mavericks v10.9 o posterior](#). La versión 2.1 y posteriores de .NET Core requieren versiones más recientes de macOS, tal y como se describe en <https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos>.

Si está utilizando .NET Core 1.0, .NET Core [usa OpenSSL en macOS](#), una dependencia que debe instalarse por separado. OpenSSL ha incluido soporte para TLS 1.2 en la versión 1.0.1 y soporte para TLS 1.3 en la versión 1.1.1.

Linux

.NET Core en Linux requiere OpenSSL, que viene incluido con muchas distribuciones de Linux. Pero también se puede instalar por separado. OpenSSL ha incluido soporte para TLS 1.2 en la versión 1.0.1 y soporte para TLS 1.3 en la versión 1.1.1. Si está utilizando una versión moderna de .NET Core (2.1 o posterior) y ha instalado un administrador de paquetes, es probable que se haya instalado una versión más moderna de OpenSSL para usted.

Para estar seguro, puede ejecutar **openssl version** en un terminal y verificar que la versión sea posterior a 1.0.1.

.NET Framework

Si está ejecutando una versión moderna de .NET Framework (4.7 o posterior) y una versión moderna de Windows (al menos Windows 8 para clientes, Windows Server 2012 o posterior para servidores), TLS 1.2 está habilitado y utilizado de forma predeterminada.

Si utiliza un entorno de ejecución de .NET Framework que no utiliza la configuración del sistema operativo (.NET Framework 3.5 a 4.5.2), AWS SDK para .NET intentará [añadir compatibilidad con TLS 1.1 y TLS 1.2 a los protocolos compatibles](#). Si está utilizando .NET Framework 3.5, esto tendrá éxito solo si se instala la revisión activa adecuada, de la siguiente manera:

- Windows 10, versión 1511, y Windows Server 2016: [KB3156421](#)
- Windows 8.1 y Windows Server 2012 R2 — [KB3154520](#)
- Windows Server 2012 — [KB3154519](#)
- Windows 7 SP1 y Server 2008 R2 SP1 — [KB3154518](#)

Warning

El 15 de agosto de 2024, dejaron de ser AWS SDK para .NET compatibles con .NET Framework 3.5 y cambiaron la versión mínima de .NET Framework a la 4.7.2. Para obtener más información, consulte la entrada del blog [Cambios importantes que se avecinan para los objetivos .NET Framework 3.5 y 4.5 de AWS SDK para .NET](#).

Si su aplicación se ejecuta en una versión más reciente de .NET Framework en Windows 7 SP1 o Windows Server 2008 R2 SP1, debe asegurarse de que la compatibilidad con TLS 1.2 esté habilitada en el registro, tal y como se describe en <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings> #tls -12. Las versiones más recientes de Windows lo tienen [habilitado de forma predeterminada](#).

Para obtener información detallada sobre las prácticas recomendadas para el uso de TLS con .NET Framework, consulte el artículo de Microsoft en <https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls>.

Herramientas de AWS para PowerShell

[Herramientas de AWS para PowerShell](#) utilice el AWS SDK para .NET para todas las llamadas a los AWS servicios. El comportamiento del entorno depende de la versión de Windows PowerShell que esté ejecutando, como se indica a continuación.

De Windows PowerShell 2.0 a 5.x

Windows PowerShell 2.0 a 5.x se ejecuta en .NET Framework. Puede comprobar qué motor de ejecución de .NET (2.0 o 4.0) se utiliza PowerShell mediante el siguiente comando.


```
$PSVersionTable.CLRVersion
```

- Al utilizar .NET Runtime 2.0, siga las instrucciones proporcionadas anteriormente con relación a AWS SDK para .NET y .NET Framework 3.5.

Warning

El 15 de agosto de 2024, dejaron de ser AWS SDK para .NET compatibles con .NET Framework 3.5 y cambiaron la versión mínima de .NET Framework a la 4.7.2. Para obtener más información, consulte la entrada del blog [Cambios importantes que se avecinan para los objetivos .NET Framework 3.5 y 4.5 de AWS SDK para .NET](#).

- Al utilizar .NET Runtime 4.0, siga las instrucciones proporcionadas anteriormente con relación a AWS SDK para .NET y .NET Framework 4+.

Windows 6.0 PowerShell

Windows PowerShell 6.0 y versiones posteriores se ejecutan en .NET Core. Puede verificar qué versión de .NET Core se está utilizando ejecutando el siguiente comando.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.FrameworkName], $true).FrameworkName
```

Siga las instrucciones proporcionadas anteriormente en relación con la AWS SDK para .NET versión correspondiente de .NET Core.

Xamarin

[Para Xamarin, consulte las instrucciones en -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) En resumen:

Para Android

- Requiere Android 5.0 o posterior.
- Propiedades del proyecto, opciones de Android: HttpClient la implementación debe estar configurada en Android y la SSL/TLS implementación en Native TLS 1.2+.

Para iOS

- Requiere iOS 7 o posterior.
- Propiedades del proyecto, compilación de iOS: HttpClient la implementación debe estar configurada como NSURLSesión.

Para macOS

- Requiere macOS 10.9 o posterior.
- Opciones de proyecto, compilación, compilación para Mac: HttpClient la implementación debe estar configurada como NSURLSesión.

Unity

Debe usar Unity 2018.2 o una versión posterior y usar el tiempo de ejecución de scripting equivalente de .NET 4.x. Puede configurarlo en los ajustes del proyecto, en la configuración o en el reproductor, tal y como se describe en <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html>. El tiempo de ejecución de secuencias de comandos equivalente a .NET 4.x permite la compatibilidad con TLS 1.2 en todas las plataformas Unity que ejecutan Mono o CPP. IL2

Navegador (para Blazor) WebAssembly

WebAssembly se ejecuta en el navegador en lugar de en el servidor y utiliza el navegador para gestionar el tráfico HTTP. Por lo tanto, la compatibilidad con TLS está determinada por la compatibilidad con el navegador.

[Blazor WebAssembly, en la versión preliminar de ASP.NET Core 3.1, solo es compatible con navegadores compatibles WebAssembly, tal y como se describe en <https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms>](#). Todos los navegadores convencionales eran compatibles con TLS 1.2 antes de admitirlo. WebAssembly Si este es el caso de su navegador, si su aplicación se ejecuta, puede comunicarse a través de TLS 1.2.

Consulte la documentación de su navegador para obtener más información y verificación.

Migración del cliente de cifrado S3 (de la V1 a la V2)

Note

Si utiliza la V2 y desea migrar a la V4, consulte [Migración de un cliente de cifrado S3 \(de V2 a V4\)](#).

En este tema se muestra cómo migrar las aplicaciones de la versión 1 (V1) del cliente de cifrado Amazon Simple Storage Service (S3) a la versión 2 (V2) y cómo garantizar la disponibilidad de las aplicaciones durante todo el proceso de migración.

Los objetos cifrados con el cliente de la versión 2 no se pueden descifrar con el cliente de la versión V1. Para facilitar la migración al nuevo cliente sin tener que volver a cifrar todos los objetos a la vez, se ha proporcionado un cliente “de transición a la versión 1”. Este cliente puede descifrar objetos cifrados tanto en la versión 1 como en la versión 2, pero solo cifra objetos en un formato compatible con la versión 1. El cliente de la versión 2 puede descifrar objetos cifrados tanto en la versión 1 como en la versión 2 (si está habilitada para objetos de la versión 1), pero solo cifra objetos en un formato compatible con la versión 2.

Información general sobre la migración

Esta migración se produce en tres fases. Estas fases se indican aquí y se describen en detalle más adelante. Cada fase debe completarse en todos los clientes que utilizan objetos compartidos para que la siguiente fase pueda iniciarse.

1. Actualizar los clientes existentes a clientes de transición a la versión 1 para leer nuevos formatos. En primer lugar, actualice las aplicaciones para que establezcan una dependencia con el cliente de transición a la versión 1 en lugar de con cliente de la versión 1. El cliente de transición a la versión 1 permite que el código existente descifre los objetos escritos por los nuevos clientes de la versión 2 y los objetos escritos en un formato compatible con la versión 1.

Note

El cliente de transición a la versión 1 se proporciona únicamente con fines de migración. Después de pasar al cliente de transición a la versión 1, proceda a actualizar el cliente de la versión 2.

2. Migrar los clientes de transición a la versión 1 a clientes de la versión 2 para escribir nuevos formatos. A continuación, reemplace todos los clientes de transición a la versión 1 de las aplicaciones por clientes de la versión 2 y establezca el perfil de seguridad en V2AndLegacy. Establecer así este perfil de seguridad en los clientes de la versión 2 permite a esos clientes descifrar los objetos que se cifraron en un formato compatible con la versión 1.
3. Actualizar los clientes de la versión 2 para que dejen de leer formatos de la versión 1. Por último, una vez que todos los clientes se hayan migrado a la versión 2 y todos los objetos se hayan cifrado o vuelto a cifrar en un formato compatible con la versión 2, establezca el perfil de seguridad de la versión 2 en V2 en lugar de en V2AndLegacy. Esto impide el descifrado de los objetos que están en un formato compatible con la versión 1.

Actualización de los clientes existentes a clientes de transición a la versión 1 para leer nuevos formatos

El cliente de cifrado de la versión 2 utiliza algoritmos de cifrado que las versiones anteriores del cliente no admiten. El primer paso de la migración consiste en actualizar los clientes de descifrado de la versión 1 para que puedan leer el nuevo formato.

El cliente de transición a la versión 1 permite a las aplicaciones descifrar los objetos cifrados tanto en la versión 1 como en la versión 2. Este cliente forma parte del paquete [Amazon.Extensions.S3.Encryption](#). NuGet Realice los siguientes pasos en cada una de sus aplicaciones para utilizar el cliente de transición a la versión 1.

1. Establezca una nueva dependencia en el paquete [Amazon.Extensions.S3.Encryption](#). Si su proyecto depende directamente del .S3 o. AWSSDK AWSSDK KeyManagementServicepaquetes, debe actualizar esas dependencias o eliminarlas para que sus versiones actualizadas se incluyan en este nuevo paquete.
2. Cambie la instrucción using apropiada de Amazon.S3.Encryption a Amazon.Extensions.S3.Encryption. Así:

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. Recompile y vuelva a implementar la aplicación.

El cliente de transición a la versión 1 es totalmente compatible con las API del cliente de la versión 1, por lo que no es necesario realizar ningún otro cambio de código.

Migración de los clientes de transición a la versión 1 a clientes de la versión 2 para escribir nuevos formatos

El cliente V2 forma parte del paquete [NuGet Amazon.Extensions.S3.Encryption](#). Permite a las aplicaciones descifrar los objetos cifrados tanto con la versión 1 como con la versión 2 (si están configuradas para ello), pero solo cifra los objetos en un formato compatible con la versión 2.

Después de actualizar los clientes existentes para leer el nuevo formato de cifrado, puede proceder a actualizar las aplicaciones sin problemas a la versión 2 de los clientes de cifrado y descifrado. Realice los siguientes pasos en cada una de las aplicaciones para utilizar el cliente de la versión 2:

1. Cambie `EncryptionMaterials` a `EncryptionMaterialsV2`.
 - a. Si usa KMS:
 - i. Proporcione un ID de clave de KMS.
 - ii. Declare el método de cifrado que está utilizando, esto es, `KmsType.KmsContext`.
 - iii. Proporcione un contexto de cifrado a KMS para asociarlo a esta clave de datos. Puede enviar un diccionario vacío (el contexto de cifrado de Amazon se seguirá combinando), pero se recomienda incluir más contexto.
 - b. Si usa métodos de empaquetado de claves proporcionados por el usuario (cifrado simétrico o asimétrico):
 - i. Proporcione una instancia de AES o de RSA que contenga los materiales de cifrado.
 - ii. Declare qué algoritmo de cifrado utilizar, esto es, `SymmetricAlgorithmType.AesGcm` o `AsymmetricAlgorithmType.RsaOaepSha1`.
2. Cambie `AmazonS3CryptoConfiguration` a `AmazonS3CryptoConfigurationV2` con la propiedad `SecurityProfile` establecida en `SecurityProfile.V2AndLegacy`.
3. Cambie `AmazonS3EncryptionClient` a `AmazonS3EncryptionClientV2`. Este cliente toma los objetos `AmazonS3CryptoConfigurationV2` y `EncryptionMaterialsV2` recién convertidos de los pasos anteriores.

Ejemplo: de KMS a KMS+Contexto

Antes de la migración

```
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);
```

Después de la migración

```
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);
```

Ejemplo: algoritmo simétrico (empaquetado de claves de AES-CBC a AES-GCM)

StorageMode puede ser ObjectMetadata o InstructionFile.

Antes de la migración

```
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);
```

Después de la migración

```
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

Al descifrar con AES-GCM, lea todo el objeto hasta el final antes de empezar a utilizar los datos descifrados. Esto se hace para verificar que el objeto no se ha modificado desde que se cifró.

Ejemplo: algoritmo asimétrico (de RSA a 1 clave) RSA-OAEP-SHA

StorageMode puede ser ObjectMetadata o InstructionFile.

Antes de la migración

```
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);
```

Después de la migración

```
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);
```

Actualización de clientes de la versión 2 para que dejen de leer formatos de la versión 1

Con el tiempo, todos los objetos acabarán cifrándose o volviéndose a cifrar con un cliente de la versión 2. Una vez completada esta conversión, puede deshabilitar la compatibilidad con la versión 1 en los clientes de la versión 2 estableciendo la propiedad `SecurityProfile` en `SecurityProfile.V2`, como se muestra en el siguiente fragmento de código.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migración de un cliente de cifrado S3 (de V2 a V4)

Note

Si utiliza la V1 y desea migrar a la V4, primero debe migrar a la V2. Consulte [Migración del cliente de cifrado S3 \(de la V1 a la V2\)](#).

En este tema se muestra cómo migrar las aplicaciones de la versión 2 (V2) del cliente de cifrado Amazon Simple Storage Service (S3) a la versión 4 (V4) y cómo garantizar la disponibilidad de

las aplicaciones durante todo el proceso de migración. La versión 4 utiliza AES-GCM con un compromiso clave para el cifrado de contenido e introduce políticas de compromiso para mejorar la seguridad contra los ataques de sustitución de claves.

[El cliente V4 está disponible en el paquete Amazon.Extensions.S3.Encryption.](#) NuGet

Important

Cambios importantes: los clientes V4 configurados con la `RequireEncryptRequireDecrypt` política no pueden descifrar los objetos cifrados con los clientes V1 o V2. Solo los clientes de la versión V2 más recientes pueden descifrar los objetos cifrados en la V4 con un compromiso clave. Antes de pasar a la `RequireEncryptRequireDecrypt` política, debe volver a cifrar todos los datos existentes mediante clientes de la versión 4 que tengan el compromiso de claves activado.

Comprender los conceptos de la versión 4

La versión 4 utiliza AES-GCM con un compromiso clave con el cifrado de contenido e introduce conceptos de seguridad clave que mejoran la protección de sus datos cifrados:

Política de compromiso

La política de compromiso controla la forma en que el cliente de cifrado gestiona el compromiso de claves durante las operaciones de cifrado y descifrado. La versión 4 admite tres políticas de compromiso:

`ForbidEncryptAllowDecrypt`

Cifrado: sin compromiso

Descifrado: permite objetos que no se comprometen

Seguridad: no impone el compromiso y puede permitir la manipulación

Compatibilidad: todas las implementaciones V2 y V4 pueden leer objetos cifrados con esta política

`RequireEncryptAllowDecrypt`

Cifrado: con un compromiso clave

Descifrado: permite confirmar y no comprometer objetos

Seguridad: los objetos nuevos están protegidos contra los ataques de sustitución de claves, y los objetos antiguos siguen siendo legibles

Compatibilidad: solo la versión 4 admite esta política

`RequireEncryptRequireDecrypt(Predeterminado para la V4)`

Cifrado: con compromiso clave

Descifrado: solo se consignan objetos

Seguridad: cumplimiento del compromiso total para una máxima seguridad

Compatibilidad: solo la versión 4 admite esta política

AES GCM con un compromiso clave

La V4 utiliza el AES-GCM con un compromiso clave para el cifrado de contenido, lo que proporciona una seguridad mejorada:

- Protección contra la manipulación: protege contra los ataques de sustitución de claves al vincular criptográficamente la clave a los datos cifrados.
- Compatibilidad de versiones: los objetos cifrados con un compromiso de clave solo pueden descifrarlos los clientes V4 y las versiones más recientes.

Warning

Antes de habilitar el cifrado por compromiso de clave en producción, asegúrese de que todas las aplicaciones que necesitan descifrar los objetos se hayan actualizado a la versión 4 o a una versión más reciente, ya que el cliente V2 está en desuso.

Actualice los clientes existentes para leer los formatos V4

El cliente de cifrado V4 utiliza algoritmos de cifrado que las versiones anteriores del cliente no admiten. El primer paso de la migración consiste en actualizar los clientes de la versión 2 para que puedan leer el nuevo formato de la versión 4.

Update NuGet Package Dependencies

Actualice sus aplicaciones para utilizar la última versión del paquete [NuGet Amazon.Extensions.S3.Encryption](#), que incluye compatibilidad con la versión 4. Realice los siguientes pasos en cada una de sus aplicaciones:

1. Actualice al paquete [Amazon.Extensions.S3.Encryption](#) más reciente. Si su proyecto depende directamente del .S3 o. AWSSDK AWSSDK KeyManagementServicepaquetes, debe actualizar esas dependencias o eliminarlas para que sus versiones actualizadas se incluyan en este nuevo paquete.
2. Asegúrese de que sus `using` declaraciones hagan referencia al espacio de nombres correcto:

```
using Amazon.Extensions.S3.Encryption;  
using Amazon.Extensions.S3.Encryption.Primitives;
```

3. Recompile y vuelva a implementar la aplicación.

Sus clientes V2 actuales seguirán trabajando con el paquete actualizado y podrán descifrar los objetos cifrados por los clientes V4 (en función de la política de compromiso utilizada).

Cree e implemente aplicaciones

Tras actualizar las dependencias de sus NuGet paquetes:

1. Cree su aplicación para asegurarse de que todas las dependencias se resuelvan correctamente.
2. Pruebe la aplicación en un entorno de desarrollo para comprobar que la funcionalidad existente sigue funcionando.
3. Implemente la aplicación actualizada en su entorno de producción.

Esta actualización permite a sus clientes de V2 existentes descifrar los objetos que serán cifrados por los clientes de V4, lo que garantiza la compatibilidad durante el proceso de migración.

Migre a clientes V4

Tras actualizar sus clientes actuales para que lean el nuevo formato de cifrado, puede proceder a actualizar sus aplicaciones de forma segura para que utilicen clientes de cifrado y descifrado V4. El cliente V4 proporciona una seguridad mejorada mediante el compromiso con las claves y, al mismo tiempo, mantiene la compatibilidad con los objetos cifrados existentes.

Proceso de migración en 4 pasos

La migración de la V2 a la V4 sigue un proceso estructurado de 4 pasos para garantizar la compatibilidad y la seguridad. Cada paso representa una configuración específica que debe implementarse en todas las aplicaciones antes de continuar con el siguiente paso.

1. Paso 0: Cliente V2 (punto de partida): su implementación V2 existente
2. Paso 1: Compatibilidad con V2: migre al cliente V4 y mantenga un comportamiento de cifrado compatible con la V2
3. Paso 2: Versión 4 con escritura con compromiso de clave: comience a cifrar con compromiso de clave y, al mismo tiempo, permita descifrar los objetos heredados
4. Paso 3: V4 con plena aplicación: se requiere el compromiso de clave tanto para el cifrado como para el descifrado

Paso 0: Cliente V2 (punto de partida)

Esto representa la configuración de su cliente V2 existente. Este paso muestra el estado inicial antes de la migración.

```
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  
    });
```

Paso 1: Compatibilidad entre V4 y V2

Migre al cliente V4 manteniendo un comportamiento idéntico al de la V2. En este paso se utiliza una `ForbidEncryptAllowDecrypt` política para cifrar sin compromiso y permitir el descifrado de todos los 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.ForbidEncryptAllowDecrypt,  
    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  
});
```

Comportamiento: cifra sin compromiso, puede descifrar tanto los objetos que se están confirmando como los que no lo están. Comportamiento idéntico al de la V2.

Paso 2: V4 con escrituras clave

Comience a cifrar con el compromiso de clave y, al mismo tiempo, mantenga la compatibilidad con versiones anteriores para el descifrado. En este paso se utiliza `RequireEncryptAllowDecrypt` la política.

⚠ Warning

Antes de implementar el paso 2, asegúrese de que todos los lectores se hayan actualizado al paso 1 o a una versión posterior para gestionar el cifrado por compromiso de clave.

```
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.RequireEncryptAllowDecrypt,
    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
});
```

Comportamiento: cifra con compromiso, puede descifrar tanto los objetos confirmados como los que no se comprometen. Los objetos nuevos están protegidos contra los ataques de sustitución de claves.

Paso 3: V4 con plena aplicación

Exija un compromiso clave tanto para el cifrado como para el descifrado. En este paso se utiliza una `RequireEncryptRequireDecrypt` política para garantizar la máxima seguridad.

⚠ Warning

Antes de implementar el paso 3, asegúrese de que todos los objetos del sistema se hayan vuelto a cifrar con la clave (paso 2). Este paso no podrá descifrar los objetos cifrados sin compromiso.

```
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.RequireEncryptRequireDecrypt,
    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
});
```

Comportamiento: cifra con compromiso, solo descifra los objetos cifrados con compromiso. Máxima seguridad contra los ataques de sustitución de claves.

Ejemplos de configuración adicionales

En esta sección se proporcionan ejemplos adicionales para configurar los clientes de la versión 4 con diferentes opciones durante la migración.

Habilitación de Legacy Support

Para permitir que los clientes V4 lean los objetos cifrados por los clientes V1 y V2, configure el cliente con una política de compromiso que permita el descifrado tradicional:

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
    CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Utilice esta configuración durante la migración cuando necesite descifrar objetos cifrados por clientes antiguos y, al mismo tiempo, garantizar que los nuevos objetos estén cifrados con mayor seguridad.

Configuración del método de almacenamiento

La versión 4 admite dos métodos de almacenamiento para los metadatos de cifrado. Elija el método que mejor se adapte a su caso de uso:

Metadatos de objetos (predeterminados)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Archivo de instrucciones

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt)
{
```



```
StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

Úselo `InstructionFile` cuando necesite conservar los metadatos de los objetos para otros fines o cuando trabaje con objetos que tienen limitaciones de tamaño de los metadatos.

Configuración de la política de compromiso

Elija la política de compromiso adecuada en función de sus requisitos de seguridad y de la fase de migración:

Fase de migración (compatibilidad con la versión 2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

Fase de transición (recomendada)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt);
```

Fase de seguridad total

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt);
```

Comience con `ForbidEncryptAllowDecrypt` la migración inicial, pase a la fase `RequireEncryptAllowDecrypt` de transición y, finalmente, utilícela `RequireEncryptRequireDecrypt` cuando todos los clientes se hayan actualizado y todos los objetos se hayan vuelto a cifrar con compromiso.

Consideraciones especiales para la AWS SDK para .NET

Esta sección contiene consideraciones relativas a casos especiales en los que las configuraciones o los procedimientos normales no son adecuados o suficientes.

Temas

- [Obtención de ensamblajes para el AWS SDK para .NET](#)
- [Acceso a las credenciales y perfiles en una aplicación](#)
- [Consideraciones especiales sobre la compatibilidad con Unity](#)
- [Consideraciones especiales sobre la compatibilidad con Xamarin](#)

Obtención de ensamblajes para el AWS SDK para .NET

En este tema se describe cómo obtener los AWSSDK ensamblajes y almacenarlos localmente (o in situ) para utilizarlos en sus proyectos. Este no es el método recomendado para administrar las referencias de SDK, pero en algunos entornos es obligatorio.

Note

El método recomendado para gestionar las referencias del SDK consiste en descargar e instalar solo los NuGet paquetes que necesite cada proyecto. Este método se describe en [Instale AWSSDK paquetes con NuGet](#).

Si no puedes o no tienes permiso para descargar e instalar NuGet paquetes por proyecto, tienes a tu disposición las siguientes opciones.

Descarga y extracción de archivos ZIP

(Recuerde que este no es el [método recomendado](#) para gestionar las referencias a AWS SDK para .NET.)

1. Descargue uno de los siguientes archivos ZIP:

- [aws-sdk-net8.0.zip](#): ensamblajes compatibles con .NET 8 y versiones posteriores.
- [aws-sdk-netcoreapp3.1.zip](#): ensamblajes compatibles con .NET Core 3.1 y versiones posteriores.

- [aws-sdk-netstandard2.0.zip](#): ensamblajes compatibles con .NET Standard 2.0 y 2.1.
 - [aws-sdk-net472.zip](#): ensamblajes compatibles con .NET Framework 4.5 y versiones posteriores.
2. Extraiga los ensamblados a una carpeta de “descargas” de su sistema de archivos; no importa dónde. Anote esta carpeta.
 3. Cuando configure el proyecto, los ensamblados necesarios se obtienen de esta carpeta, tal y como se describe en [Instale AWSSDK conjuntos sin NuGet](#).

Acceso a las credenciales y perfiles en una aplicación

El método preferido para usar las credenciales es permitir que las AWS SDK para .NET encuentren y las recuperen por usted, tal y como se describe en [Resolución de credencial y perfil](#).

Sin embargo, también puede configurar la aplicación para que recupere perfiles y credenciales de forma activa y, a continuación, utilizarlas de forma explícita al crear un cliente de AWS servicio.

Para recuperar perfiles y credenciales de forma activa, utilice las clases de [Amazon.Runtime.CredentialManagement](#) espacio de nombres.

- Para buscar un perfil en un archivo que utilice el formato de archivo de AWS credenciales (el [archivo de AWS credenciales compartido en su ubicación predeterminada](#) o un archivo de credenciales personalizado), utilice la [SharedCredentialsFile](#) clase. En aras de una mayor brevedad, en este texto los archivos con este formato a veces se denominarán simplemente archivos de credenciales.
- Para buscar un perfil en la tienda del SDK, usa la clase [Net SDKCredentials File](#).
- Para buscar tanto en un archivo de credenciales como en la tienda del SDK, en función de la configuración de una propiedad de clase, usa la [CredentialProfileStoreChain](#) clase.

Esta clase se puede usar para encontrar perfiles. También puedes usar esta clase para solicitar AWS credenciales directamente en lugar de usar la [AWSCredentialsFactory](#) clase (que se describe a continuación).

- Para recuperar o crear varios tipos de credenciales a partir de un perfil, utilice la clase [AWSCredentialsFactory](#).

En las siguientes secciones se muestran ejemplos de estas clases.

Ejemplos de clase CredentialProfileStoreChain

Puede obtener credenciales o perfiles de la [CredentialProfileStoreChain](#) clase mediante los [TryGetProfile](#) métodos [TryGetAWSCredentials](#). La propiedad `ProfilesLocation` de la clase determina el comportamiento de los métodos del siguiente modo:

- Si `ProfilesLocation` es nulo o está vacío, busca en la tienda del SDK si la plataforma lo admite y, a continuación, busca el archivo de AWS credenciales compartidas en la ubicación predeterminada.
- Si la propiedad `ProfilesLocation` contiene un valor, busque en el archivo de credenciales especificado en la propiedad.

Obtenga las credenciales de la tienda del SDK o del archivo de AWS credenciales compartidas

En este ejemplo se muestra cómo obtener credenciales mediante la clase `CredentialProfileStoreChain` y cómo usarlas a continuación para crear un objeto [AmazonS3Client](#). Las credenciales pueden proceder de SDK Store o del archivo de credenciales de AWS compartido en la ubicación predeterminada.

En este ejemplo también se usa [Amazon.Runtime.AWSCredentials](#) clase.

```
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}");
    }
}
```

Obtenga un perfil de la tienda del SDK o del archivo de AWS credenciales compartido

En este ejemplo, se muestra cómo obtener un perfil mediante la `CredentialProfileStoreChain` clase. Las credenciales pueden provenir de la tienda del SDK o del archivo de AWS credenciales compartido en la ubicación predeterminada.

En este ejemplo también se usa la [CredentialProfile](#) clase.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Obtención de credenciales desde un archivo de credenciales personalizado

En este ejemplo, se muestra cómo obtener credenciales mediante la `CredentialProfileStoreChain` clase. Las credenciales provienen de un archivo que usa el formato de archivo de AWS credenciales, pero se encuentra en una ubicación alternativa.

En este ejemplo también se utiliza [Amazon.Runtime.AWSCredentials](#) clase.

```
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
}
```

Ejemplos de clases `SharedCredentialsFile` y `AWSCredentials Factory`

Cree un `AmazonS3Client` mediante la clase `SharedCredentialsFile`

[En este ejemplo, se muestra cómo buscar un perfil en el archivo de AWS credenciales compartido, crear AWS credenciales a partir del perfil y, a continuación, utilizar las credenciales para crear un objeto `AmazonS3Client`.](#) En el ejemplo se usa la clase. [SharedCredentialsFile](#)

En este ejemplo también se utiliza la [CredentialProfile](#) clase y el [Amazon.Runtime.AWSCredentials](#) clase.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
    awsCredentials))
```

```
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Note

La clase [Net SDKCredentials File](#) se puede usar exactamente de la misma manera, excepto que se crea una instancia de un nuevo objeto Net SDKCredentials File en lugar de un SharedCredentialsFile objeto.

Consideraciones especiales sobre la compatibilidad con Unity

Cuando utilice el [estándar 2.0 AWS SDK para .NET y .NET](#) para su aplicación Unity, su aplicación debe hacer referencia a los AWS SDK para .NET ensamblados (archivos DLL) directamente en lugar de usarlos NuGet. Debido a este requisito, estas son algunas acciones importantes que deberá realizar.

- Debe obtener los AWS SDK para .NET ensamblajes y aplicarlos a su proyecto. Para obtener información sobre cómo hacerlo, consulte [Descarga y extracción de archivos ZIP](#) en el tema [Obtención de AWSSDK conjuntos](#).
- Debes incluir lo siguiente DLLs en tu proyecto de Unity junto con el DLLs de AWSSDK.Core y los demás AWS servicios que utilices. A partir de la versión 3.5.109 del AWS SDK para .NET, el archivo ZIP de .NET Standard contiene estos elementos adicionales. DLLs
 - [Microsoft.Bcl.AsyncInterfaces.dll](#)
 - [System.Runtime.CompilerServices.Unsafe.dll](#)
 - [System.Threading.Tasks.Extensions.dll](#)
- Si utilizas el [IL2CPP](#) para crear tu proyecto de Unity, debes añadir un link.xml archivo a la carpeta Asset para evitar la extracción de código. El link.xml archivo debe enumerar todos los

AWSSDK ensamblajes que estés utilizando y cada uno debe incluir el `preserve="all"` atributo. El siguiente fragmento de código es un ejemplo de este archivo.

```
<linker>
  <assembly fullname="AWSSDK.Core" preserve="all"/>
  <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
  <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

Para leer información básica interesante relacionada con este requisito, consulte el artículo en <https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-uwp/>. from-unity-xamarin-or

Consideraciones especiales sobre la compatibilidad con Xamarin

Los proyectos de Xamarin (nuevos y existentes) deben dirigirse a .NET Standard 2.0. Consulte [Compatibilidad de .NET Standard 2.0 en Xamarin.Forms](#) y [Compatibilidad con implementaciones de .NET](#).

Consulte también la información en [Biblioteca de clases portátil y Xamarin](#).

Referencia de API para AWS SDK para .NET

AWS SDK para .NET Proporciona una API que puede utilizar para acceder a AWS los servicios. Para ver qué clases y métodos están disponibles en la API, consulte la [Referencia de API de AWS SDK para .NET](#).

Además de la referencia general indicada anteriormente, cada uno de los ejemplos de la sección [Ejemplos de código guiados](#) contiene referencias a los métodos y clases específicos que se utilizan en ese ejemplo.

Acerca de las versiones de referencia de la API

La referencia de la API descrita anteriormente corresponde a la versión 3.0 y posteriores de AWS SDK para .NET.

Para obtener información sobre la migración desde versiones anteriores del SDK, consulte [Migración de su proyecto](#)

Para encontrar contenido obsoleto de las versiones anteriores de la referencia de la API del SDK, consulta los siguientes elementos:

- [AWS SDK para .NET Referencia de API V1 \(obsoleta\)](#)
- [AWS SDK para .NET Referencia de API V2 \(obsoleta\)](#)

Para ver una referencia de AWS SDK para .NET API obsoleta, tendrás que extraerla y configurar un navegador web. Las instrucciones que se muestran a continuación son ejemplos de cómo hacerlo. Se basan en el uso del navegador web Google Chrome en un sistema Windows. Adáptelos a su navegador web y sistema operativo específicos.

Consulta la versión 1 de referencia de la API, que está obsoleta

1. Descarga el archivo ZIP de la AWS SDK para .NET API de referencia V1, que está en desuso. De forma predeterminada, el nombre del archivo ZIP `essdkfornet-api-ref_v1_deprecated.zip`. Ese nombre se utilizará a lo largo de estas instrucciones.
2. Coloque el archivo ZIP en la carpeta que desee. Para estas instrucciones, se supone que el nombre de la carpeta es `C:\work\temp\api-refs\V1`.

3. Haga clic con el botón derecho en el archivo ZIP y seleccione Extraer todo. Acepte la ubicación por defecto, que es C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1 la de estas instrucciones.
4. Crea un acceso directo para Google Chrome en la C:\work\temp\api-refs\V1 carpeta. Tenga cuidado de no mover la aplicación original.
5. En las propiedades del nuevo acceso directo, defina los siguientes campos:
 - Objetivo: "*<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 el --user-data-dir argumento, utilice un nombre de carpeta que funcione para su entorno. La carpeta no tiene por qué existir.

 - Comience en: C:\work\temp\api-refs\V1
6. Asigne al acceso directo un nombre apropiado.
7. Abre el acceso directo para ver la referencia de la API anterior.

Consulta la versión 2 de referencia de la API, que está obsoleta

1. Descarga el archivo ZIP de la AWS SDK para .NET API Reference V2, que está obsoleta. De forma predeterminada, el nombre del archivo ZIP es `essdkfornet-api-ref_v2_deprecated.zip`. Ese nombre se utilizará a lo largo de estas instrucciones.
2. Coloque el archivo ZIP en la carpeta que desee. Para estas instrucciones, se supone que el nombre de la carpeta es C:\work\temp\api-refs\V2.
3. Haga clic con el botón derecho en el archivo ZIP y seleccione Extraer todo. Acepte la ubicación por defecto, que es C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2 la de estas instrucciones.
4. Crea un acceso directo para Google Chrome en la C:\work\temp\api-refs\V2 carpeta. Tenga cuidado de no mover la aplicación original.
5. En las propiedades del nuevo acceso directo, defina los siguientes campos:
 - Objetivo: "*<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 el `--user-data-dir` argumento, utilice un nombre de carpeta que funcione para su entorno. La carpeta no tiene por qué existir.

- Comience en: `C:\work\temp\api-refs\V2`
6. Asigne al acceso directo un nombre apropiado.
 7. Abre el acceso directo para ver la referencia de la API anterior.

Historial de documentos

En la siguiente tabla se describen los cambios importantes que se han realizado en la documentación desde la última versión de la Guía para desarrolladores de AWS SDK para .NET . Para obtener notificaciones sobre las actualizaciones de esta documentación, puede suscribirse a una [fuente RSS](#).

Cambio	Descripción	Fecha
Novedades	Se agregó una nota sobre la versión 2.0 de la herramienta de AWS implementación para .NET CLI.	17 de octubre de 2025
AWS OpsWorks	AWS OpsWorks ha llegado al final de su vida útil. Se ha eliminado de la documentación.	16 de septiembre de 2025
Novedades	End-of-support ha sido anunciado para la versión 3 del AWS SDK para .NET.	21 de agosto de 2025
Novedades	Disponibilidad general para el proveedor de caché distribuida	2 de julio de 2025
Marco de procesamiento de mensajes	¡El marco de procesamiento de AWS mensajes para.NET está disponible para el público en general!	5 de mayo de 2025
Novedades	¡La versión 4 AWS SDK para .NET está disponible de forma general! Para obtener información sobre la migración de las aplicaciones a la V4,	28 de abril de 2025

consulte [Migración a la versión 4](#).

[Migración a la versión 4 de AWS SDK para .NET](#)

Se incluyó información preliminar sobre la migración de las aplicaciones creadas con la versión AWS SDK para .NET 4.

3 de marzo de 2025

[Integración AWS con .NET Aspire en la AWS SDK para .NET](#)

Se incluyó información sobre cómo se AWS ha integrado con .NET Aspire en el AWS SDK para .NET.

15 de febrero de 2025

[Observabilidad](#)

Se ha anunciado la disponibilidad general para la observabilidad.

10 de febrero de 2025

[Novedades](#)

Se ha agregado información sobre el nuevo comportamiento predeterminado para la protección de la integridad.

15 de enero de 2025

[Migración a S3 Encryption Client V4](#)

Se agregó información sobre la migración de la V2 a la V4.

9 de diciembre de 2024

[Novedades](#)

Se agregó información sobre la cuarta versión preliminar de la AWS SDK para .NET versión 4.

15 de noviembre de 2024