Guía para desarrolladores

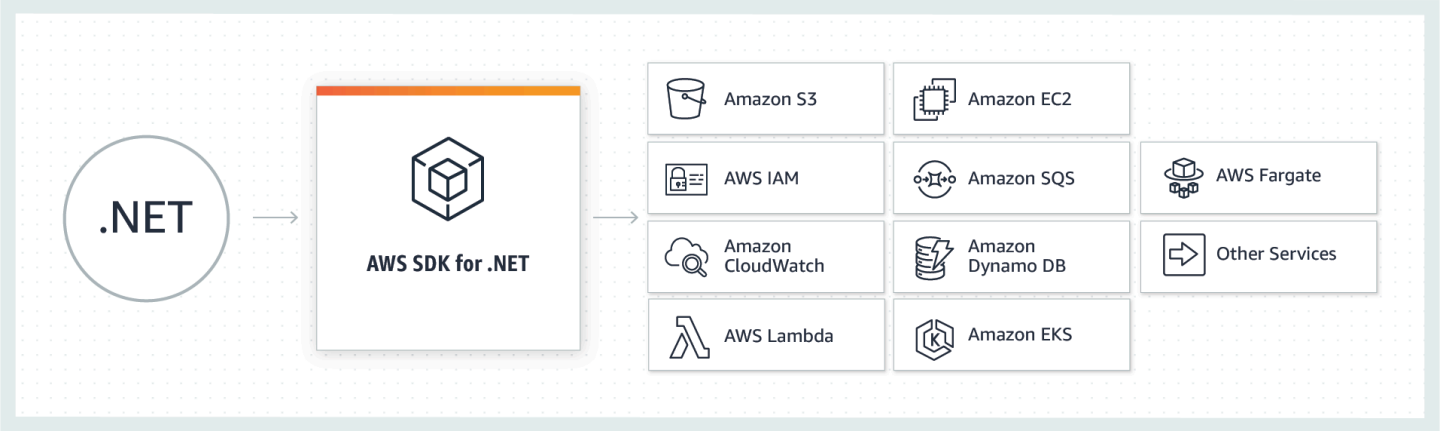
AWS SDK for .NET

AWS SDK for .NET: Guía para desarrolladores

Table of Contents

¿Qué es el AWS SDK for .NET

AWS SDK for .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). El SDK simplifica el uso de los AWS servicios al proporcionar un conjunto de bibliotecas coherentes y familiares para los desarrolladores de .NET.



(¡Está bien, lo tengo! Estoy listo para prepararme y hacer un recorrido rápido).

Acerca de esta versión

> **Note**
>
> Esta documentación corresponde a la versión 3.0 y posteriores de AWS SDK for .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, consulte Migración del proyecto.
> Para encontrar contenido obsoleto de versiones anteriores de AWS SDK for .NET, consulte los siguientes elementos:
>
> - AWS SDK for .NET (versión 2, obsoleta) Guía para desarrolladores

Mantenimiento y compatibilidad de las versiones principales del SDK

Para obtener información sobre el mantenimiento y la compatibilidad con las principales versiones del SDK y sus dependencias subyacentes, consulte lo siguiente en la Guía de Referencia de SDK y herramientas de AWS:

- AWS Política de mantenimiento de SDK y herramientas
- AWS Matriz de soporte de versiones de SDK y herramientas

Casos de uso comunes

AWS SDK for .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
- Creación y lanzamiento de instancias de Amazon EC2 y configuración y solicitud de instancias de spot de Amazon EC2

Temas adicionales en esta sección

- Herramientas de AWS relacionadas con AWS SDK for .NET
- Guía de referencia de SDK y herramientas de AWS
- Recursos adicionales

Herramientas de AWS relacionadas con AWS SDK for .NET

Herramientas para Windows PowerShell o Herramientas para PowerShell Core

AWS Tools for Windows PowerShell y AWS Tools for PowerShell Core son módulos de PowerShell basados en la funcionalidad expuesta por el AWS SDK for .NET. Con Herramientas de AWS para PowerShell puede llevar a cabo operaciones mediante script en los recursos de AWS desde un símbolo del sistema de PowerShell. Aunque los cmdlets se implementan mediante los clientes del servicio y los métodos del SDK, los cmdlets proporcionan una experiencia de PowerShell idiomático para especificar parámetros y administrar 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:

- Crear aplicaciones sin servidor que contengan funciones AWS Lambda e implementar las aplicaciones en una pila de AWS CloudFormation.
- Trabajar con esquemas de Amazon EventBridge.
- Usar IntelliSense cuando trabaje con archivos de definición de tareas de Amazon ECS.
- Ver una aplicación de AWS Cloud Development Kit (AWS CDK).

Kit de herramientas para Visual Studio

El 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 Explorador de AWS para administrar instancias de Amazon EC2, trabajar con tablas de Amazon DynamoDB, publicar mensajes en colas de Amazon Simple Notification Service (Amazon SNS), etc., todo ello dentro de Visual Studio.

Para empezar, consulte Configuración del AWS Toolkit for Visual Studio.

Kit de herramientas para Azure DevOps

El AWS Toolkit for Microsoft Azure DevOps agrega tareas para permitir fácilmente la creación y publicación de canalizaciones en Azure DevOps y Azure DevOps Server para trabajar con los servicios de AWS. Puede trabajar con Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, Lambda, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) y Amazon SNS. También puede ejecutar comandos con el módulo Windows PowerShell y la AWS Command Line Interface (AWS CLI).

Para empezar a usar el AWS Toolkit for Azure DevOps, consulte la Guía del usuario del AWS Toolkit for Microsoft Azure DevOps.

Guía de referencia de SDK y herramientas de AWS

La Guía de referencia de SDK y herramientas de AWS contiene información relevante e importante sobre muchos de los SDK y kits de herramientas de AWS y sobre 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 cuentas, usuarios y roles de AWS
- Referencia de ajustes de configuración y autenticación
- Bibliotecas Common Runtime (CRT) de AWS
- Política de mantenimiento de SDK y herramientas AWS
- Matriz de compatibilidad para versiones de SDK y herramientas AWS

Recursos adicionales

Servicios admitidos

El AWS SDK for .NET admite la mayoría de los productos de infraestructura de AWS y más servicios se añaden con frecuencia. Para obtener una lista de los servicios de AWS compatibles con el SDK, consulte el archivo README del SDK.

Historial de revisiones

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

- Registro de cambios de SDK
- Qué hay de nuevo en el AWS SDK for .NET
- Historial de documentos

Página de inicio de AWS SDK for .NET

Para obtener más información sobre AWS SDK for .NET, vaya a la página de inicio de 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 for .NET.

AWS re:post (antes, foros de AWS)

Visite AWS re:post, específicamente el tema sobre AWS SDK for .NET para hacer preguntas o realizar comentarios sobre 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 un tema de AWS SDK for .NET, vaya a la página Todos los temas, busque ".NET en AWS" y seleccione 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 repositorios aws/dotnet y aws/aws-sdk-net en el sitio web de GitHub para obtener enlaces a bibliotecas, herramientas y recursos que puede utilizar como ayuda para crear aplicaciones y servicios .NET en AWS.

A continuación se muestran algunos ejemplos:

- Extensión de configuración de AWS .NET para Systems Manager
- Extensiones de AWS para la configuración de .NET Core
- Registros de AWS para .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 AWS Cloud: Implementación de referencia de inicio rápido
- Blog Hello, Cloud!
- Documento técnico sobre AWS: Desarrollo e implementación de aplicaciones .NET en AWS
- AWS Microservice Extractor for .NET
- Asistente de portabilidad para .NET
- Guía de referencia de las herramientas y los SDK de AWS

Comenzar a utilizar AWS SDK for JavaScript

Para usar AWS SDK for .NET, debe instalar su cadena de herramientas y configurar una serie de elementos esenciales que la aplicación necesita para acceder a servicios de AWS. 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
- Especificación de la región de AWS
- Paquetes o ensamblados de AWSSDK

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

Instalación y configuración de la cadena de herramientas

Para utilizar AWS SDK for .NET, debe tener instaladas algunas herramientas de desarrollo.

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.

  Entre los ejemplos se incluyen Microsoft Visual Studio Code (VS Code), JetBrains Rider y Microsoft Visual Studio.

- (Opcional) Un kit de herramientas de AWS, si hay uno disponible para el editor escogido y para 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) AWS Toolkit for Visual Studio, que es un complemento que proporciona una interfaz de usuario para administrar los perfiles locales y recursos de AWS 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

Configuración de la autenticación de SDK con AWS

Configuración de la autenticación de SDK con AWS

Debe establecer cómo se autentica el código con AWS cuando se desarrolla con servicios de AWS. Existen diferentes formas de configurar el acceso programático a los recursos de AWS, en función del entorno y el acceso a AWS del que disponga.

Para ver varios métodos de autenticación de SDK, consulte Autenticación y acceso en la Guía de referencia de herramientas y AWS SDK.

En este tema se da por sentado que un nuevo usuario desarrolla su actividad a nivel local, que el empleador no le ha proporcionado un método de autenticación y que 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. Habilitar y configurar el Centro de identidades de IAM
2. Configuración de SDK para usar IAM Identity Center
3. Iniciar una sesión en el portal de acceso a AWS

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 hacer esto para SDK, consulte el Paso 1 de Autenticación de IAM Identity Center en la Guía de referencia de herramientas y AWS SDK. 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

Encontrará información sobre cómo configurar SDK para usar IAM Identity Center en el Paso 2 de Autenticación de IAM Identity Center en la Guía de referencia de herramientas y AWS SDK. Tras completar esta configuración, el sistema debe contener los siguientes elementos:

- La AWS CLI, que se utiliza para iniciar una sesión en el portal de acceso a AWS antes de ejecutar la aplicación.
- El archivo `config` de AWS compartido contiene un perfil `[default]` con un conjunto de valores de configuración a los que se puede hacer referencia desde SDK. Para encontrar la ubicación de este archivo, consulte Ubicación de los archivos compartidos en la Guía de referencia de herramientas y AWS SDK. AWS SDK for .NET utiliza el proveedor de token de SSO del perfil para obtener credenciales antes de enviar 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 sección `sso-session` contiene la configuración para iniciar una sesión en el portal de acceso a AWS.

```
[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 utiliza AWS IAM Identity Center para la autenticación, la aplicación debe hacer referencia a los siguientes paquetes NuGet para que la resolución de SSO funcione:

- `AWSSDK.SSO`
- `AWSSDK.SSOOIDC`

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

Iniciar una sesión en el portal de acceso a AWS

Antes de ejecutar una aplicación que accede a servicios de AWS, necesita una sesión activa en el portal de acceso a AWS para que SDK utilice la autenticación del IAM Identity Center para 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 acceso a AWS, ejecute el siguiente comando en la 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 comando de la AWS CLI.

```
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 acceso a AWS y ejecuta `aws sso login`, no tendrá que proporcionar credenciales.
> Es posible que el proceso de inicio de sesión le permita el acceso de la AWS CLI a los datos. Como la AWS CLI se ha creado con el SDK para Python, los mensajes de permiso pueden contener variaciones del nombre `botocore`.

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 SDK, como el uso de perfiles y variables de entorno, consulte el capítulo de configuración de la Guía de referencia de herramientas y AWS SDK.
- Para obtener más información sobre las prácticas recomendadas, consulte Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.
- Para crear credenciales de AWS 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 y AWS SDK.

Recorrido rápido por AWS SDK for .NET

Esta sección proporciona tutoriales básicos para los desarrolladores que no están familiarizados con AWS SDK for .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 acerca del desarrollo de software para servicios de AWS específicos junto con los ejemplo de código, consulte Trabaje con AWS los servicios. Para ver otros ejemplos de código, consulte AWS SDK for .NET ejemplos de código.

Temas

- Aplicación multiplataforma sencilla que utiliza la herramienta AWS SDK for .NET
- Aplicación sencilla basada en Windows que utiliza AWS SDK for .NET
- Pasos siguientes

Aplicación multiplataforma sencilla que utiliza la herramienta AWS SDK for .NET

En este tutorial se utiliza AWS SDK for .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.

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.

  Entre los ejemplos se incluyen 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
- Creación del 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.SSOOIDC
```

Con los comandos anteriores se instalan los paquetes Nuget desde el Administrador de paquetes NuGet. Como sabemos exactamente qué paquetes NuGet necesitamos en 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.

Creación del 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.SSOOIDC
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. Use “dotnet-quicktour-s3-1-cross-” como base y agréguele algo único, como un GUID o su 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, reemplazando *BUCKET-NAME* por el nombre del bucket elegido.

```
dotnet run BUCKET-NAME
```

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 for .NET

En este tutorial se utiliza AWS SDK for .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.

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.SSOOIDC`

   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.SSOOIDC
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, reemplace *BUCKET-NAME* en el siguiente comando por el nombre del bucket que ha elegido.

   ```
   S3CreateAndList BUCKET-NAME
   ```

   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 siguientes

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

Ahora que ya se ha paseado por AWS SDK for .NET, probablemente quiera iniciar un proyecto.

Inicio de un nuevo proyecto

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

- Si no tiene experiencia en el desarrollo de .NET en AWS o no está familiarizado con AWS SDK for .NET, puede ver ejemplos completos en Recorrido rápido, 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 agregar código y paquetes NuGet. 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 proyecto de C# para una serie de servicios de AWS. Después de instalar el kit de herramientas en Visual Studio, puede acceder a las plantillas mientras crea un nuevo proyecto.

  Para saber cómo, vaya a Uso de los servicios de AWS en la Guía del usuario de AWS Toolkit for Visual Studio. Varios de los ejemplos de esa sección sirven para crear nuevos proyectos.

- Si desarrolla con Visual Studio en Windows, pero sin 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 agregar código y paquetes NuGet.

Encontrará algunos ejemplos de cómo trabajar con servicios de AWS en Ejemplos de código con orientaciones.

> **Important**
>
> Si utiliza AWS IAM Identity Center para la autenticación, la aplicación debe hacer referencia a los siguientes paquetes NuGet para que la resolución de SSO funcione:
>
> - `AWSSDK.SSO`

- `AWSSDK.SSOOIDC`

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

Configuración de la región de AWS

Las regiones de AWS permiten acceder a servicios de AWS que se encuentran físicamente en una región geográfica determinada. 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 de cada servicio de AWS, consulte Puntos de conexión y cuotas de 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 servicio de AWS que vaya a una región en particular. También puede configurar la aplicación con una región que se utilizará en todos los clientes de servicio de AWS. Estos dos casos se explican a continuación.

Creación de un cliente de servicio con una región particular

Se puede especificar la región de cualquiera de los clientes de servicio de AWS de la 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 cliente de Amazon EC2 en una región existente. Se utilizan campos de RegionEndpoint definidos.

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

Nueva región mediante la clase RegionEndpoint

En este ejemplo se muestra cómo crear un nuevo punto de conexión 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 propiedad `ServiceURL` de la clase de configuración de cliente de servicio para especificar la región, en este caso, la clase AmazonEC2Config.

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

AWS SDK for .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 esta variable de entorno se establece para todo el sistema (mediante `export` o `setx`), afectará a todos los SDK y kits de herramientas, no solo a AWS SDK for .NET.

Clase AWSConfigs

Establézcala como una propiedad AWSConfigs.AWSRegion.

```
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 for .NET intenta encontrar una región en la que pueda operar el cliente de servicio de AWS.

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 la instancia de Amazon EC2 (si se ejecuta en una instancia de EC2)

Si no se encuentra ninguna región, SDK lanza una excepción que indica que el cliente de servicio de AWS 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 regiones de AWS no funcionarán en la región China (Pekín). De igual modo, las cuentas y credenciales de la región China (Pekín) no funcionarán en otras regiones de AWS. 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 servicios de AWS

Los servicios nuevos de AWS se pueden lanzar inicialmente en algunas regiones y pasar a ser compatibles más adelante 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.

Instalación de paquetes de AWSSDK con NuGet

NuGet es un sistema de administración de paquetes de la plataforma .NET. Con NuGet, se pueden instalar paquetes de AWSSDK, así como otras extensiones diversas, en el proyecto. Para obtener más información, consulte el repositorio aws/dotnet en el sitio web de GitHub.

NuGet siempre tiene las versiones más recientes de los paquetes de AWSSDK, así como las versiones anteriores. NuGet tiene en cuenta las dependencias entre paquetes e instala todos los paquetes necesarios de forma automática.

> **Warning**
>
> La lista de paquetes NuGet puede incluir uno llamado simplemente “AWSSDK” (sin identificador anexo). NO instale este paquete NuGet; es heredado y no debe usarse en los proyectos nuevos.

Los paquetes instalados con NuGet se almacenan con su 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 acerca de NuGet, consulte la documentación de NuGet.

> **Note**
>
> Si no puede o no tiene permiso para descargar e instalar paquetes NuGet en cada proyecto, puede obtener los ensamblados de AWSSDK y almacenarlos localmente (o en las instalaciones).
> Si es su caso y aún no ha obtenido los ensamblados de AWSSDK, consulte Obtención de AWSSDK conjuntos. Para obtener información sobre cómo utilizar los ensamblados almacenados localmente, consulte Instalación de ensamblados de AWSSDK sin NuGet.

Uso de NuGet desde la línea de comandos o un terminal

1. Acceda a los paquetes de AWSSDK en NuGet y determine qué paquetes necesita en el 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 de la CLI de .NET. NuGet también instala cualquier dependencia, como por ejemplo AWSSDK.Core.

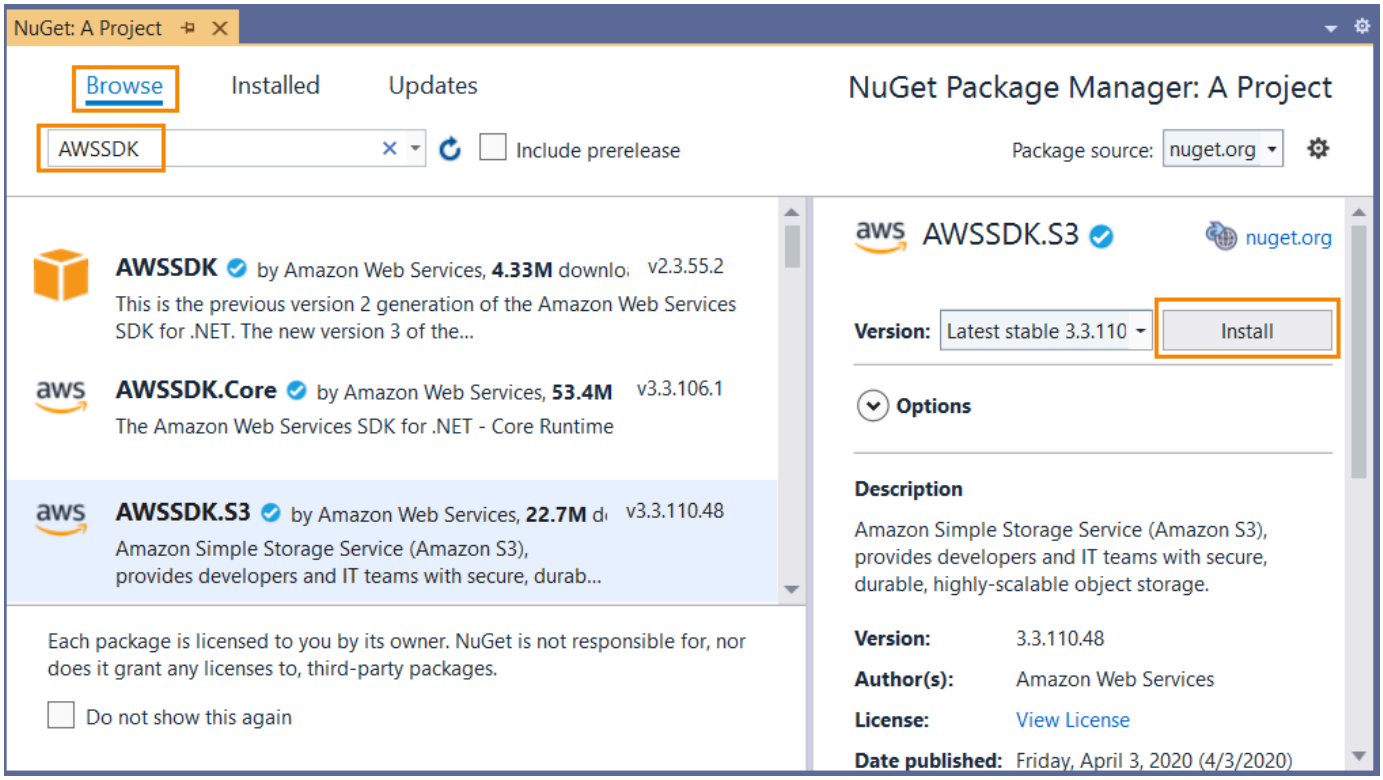
> **Note**
>
> Si solo desea la última versión de un paquete NuGet, puede excluir del comando la información de versión, como se muestra en el siguiente ejemplo.

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

Uso de NuGet 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, elija Administrar paquetes NuGet en el menú contextual.
2. En el panel izquierdo del Administrador de paquetes NuGet, elija Examinar. A continuación, puede utilizar el cuadro de búsqueda para buscar el paquete que desea instalar. NuGet también instala cualquier dependencia, como por ejemplo AWSSDK.Core.

   En la siguiente imagen se muestra la instalación del paquete AWSSDK.S3.



Uso de NuGet desde la Consola del administrador de paquetes

En Visual Studio, seleccione Herramientas > Administrador de paquetes NuGet > Consola del Administrador de paquetes.

Puede instalar los paquetes de AWSSDK que desee desde la consola del Administrador de paquetes utilizando el comando **`Install-Package`**. Por ejemplo, para instalar AWSSDK.S3, utilice el siguiente comando.

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

NuGet también instala cualquier dependencia, como por ejemplo AWSSDK.Core.

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 la consola del Administrador de paquetes, consulte la referencia de PowerShell en la documentación de NuGet de Microsoft.

Instalación de ensamblados de AWSSDK sin NuGet

En este tema se describe cómo puede utilizar los ensamblados de AWSSDK que ha obtenido y ha almacenado localmente, 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 administrar las referencias de SDK es descargar e instalar solo los paquetes NuGet que se necesiten en cada proyecto. Este método se describe en Instalación de paquetes de AWSSDK con NuGet.

Para instalar ensamblados de AWSSDK

1. Cree una carpeta en el área del proyecto de ensamblados de AWSSDK necesarios. A modo de ejemplo, podría denominar esta carpeta `AwsAssemblies`
2. Si aún no lo ha hecho, obtenga los ensamblados de AWSSDK, que se colocarán en una carpeta local de descargas 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. Encontrará información sobre las dependencias en el sitio web de GitHub.

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.

Resolución de credencial y perfil

AWS SDK for .NET busca credenciales en un determinado orden y usa el primer conjunto disponible para la aplicación actual.

Orden de búsqueda de credenciales

1. Credenciales establecidas de forma explícita en el cliente de servicio de AWS, 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. Un perfil de credenciales con el nombre especificado por un valor en AWSConfigs.AWSProfileName.
3. Un perfil de credenciales con el nombre especificado por la variable de entorno `AWS_PROFILE`.
4. El perfil de credenciales `[default]`.
5. SessionAWSCredentials creadas a partir de las variables de entorno `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` y `AWS_SESSION_TOKEN`, si no están todas vacías.
6. BasicAWSCredentials creadas a partir de las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY`, si ambas no están vacías.
7. Roles de IAM para tareas de Amazon ECS
8. Metadatos de la instancia de Amazon EC2

Si la aplicación se ejecuta en una instancia de Amazon EC2, como un entorno de producción, utilice un rol de IAM tal como se describe en Concesión de acceso mediante un rol de IAM. Si no es el caso (como, por ejemplo, en las pruebas de versión preliminar) almacene las credenciales en un archivo de credenciales de AWS al que la aplicación web tenga acceso en el servidor.

Resolución de perfiles

Con dos tipos de mecanismos de almacenamiento de credenciales diferentes, es importante saber cómo configurar AWS SDK for .NET para usarlos. La propiedad AWSConfigs.AWSProfilesLocation controla la forma en que AWS SDK for .NET busca 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 `~/.aws/config` (Linux o macOS) o en `%USERPROFILE%\.aws\config` (Windows).
Ruta a un archivo en el formato del archivo de credenciales de AWS	Busque solo en el archivo especificado el perfil con el nombre indicado.

Uso de credenciales de cuentas de usuario federado

Las aplicaciones que usan AWS SDK for .NET (AWSSDK.Core versión 3.1.6.0 y posterior) pueden usar cuentas de usuario federado a través de Servicios de federación de Active Directory (AD FS) para obtener acceso a servicios de AWS mediante el lenguaje de marcado de aserción 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 usan cuando la aplicación invoca servicios de 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 (para las cuentas que no unidas a un dominio, se le pide al usuario que escriba las credenciales antes de la segunda autenticación).

Para usar este soporte en su aplicación de .NET, primero debe configurar el perfil de rol mediante un cmdlet de PowerShell. 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. Hay varias formas de hacerlo; una de ellas consiste en utilizar la propiedad AWSConfigs.AWSProfileName del mismo modo que se utilizaría con otros perfiles de credenciales.

El ensamblado de AWS Security Token Service (AWSSDK.SecurityToken) proporciona la compatibilidad con SAML para obtener credenciales de AWS. 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 instancias de Amazon EC2, el modo más seguro de administrar las credenciales es usar roles de IAM, 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 acceso restringido a los recursos de AWS, estas credenciales tienen la ventaja de caducar después de un período de tiempo especificado. 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 el software se comunica con AWS a través de un proxy, puede especificar credenciales para el proxy mediante la propiedad `ProxyCredentials` en la clase `Config` de un servicio. La clase `Config` de un servicio suele formar parte del espacio de nombres principal del servicio. Ejemplos de ello son los siguientes AmazonCloudDirectoryConfig del espacio de nombres Amazon.CloudDirectory y AmazonGameLiftConfig del espacio de nombres Amazon.GameLift.

Por ejemplo, en Amazon S3 se podría usar un código similar al siguiente, donde `SecurelyStoredUserName` y `SecurelyStoredPassword` son el nombre de usuario y la contraseña del proxy especificados en un objeto NetworkCredential.

```
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.

Información adicional acerca de los usuarios y los roles

Para desarrollar con .NET en AWS o ejecutar aplicaciones .NET en AWS, debe tener una combinación de usuarios, conjuntos de permisos y roles de servicio que sean adecuados 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 en AWS IAM Identity Center y utilizar credenciales temporales proporcionadas por un origen de identidad.

Para el desarrollo, puede usar el usuario que ha creado o que le han proporcionado en Configuración de la autenticación de SDK. Si tiene los permisos de AWS Management Console adecuados, también puede crear diferentes conjuntos de permisos con los privilegios mínimos para ese usuario o crear nuevos usuarios específicamente para proyectos de desarrollo, lo que proporciona 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 Guía de referencia herramientas y AWS SDK e Introducción en la Guía del usuario de AWS IAM Identity Center.

Roles de servicio

Puede configurar un rol de servicio de AWS para acceder a los servicios de AWS en nombre de los usuarios. Este tipo de acceso es adecuado si varias personas van a ejecutar la aplicación de forma remota; por ejemplo, en una instancia de Amazon EC2 que haya 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 AWS Management Console y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. Elija Roles y después Crear rol.
3. Elija el servicio de AWS, busque y seleccione EC2 (por ejemplo) y, a continuación, elija el caso de uso de EC2 (por ejemplo).
4. Seleccione Siguiente: Permisos y seleccione las políticas adecuadas para los servicios de AWS que la aplicación va a utilizar.

   > ⚠ Warning
   >
   > NO elija la política AdministratorAccess porque esa política permite permisos de lectura y escritura en casi todo el contenido de la cuenta.

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

   Encontrará información sobre las etiquetas en Control de acceso a los recursos de AWS mediante etiquetas 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 Identidades 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).
- Use roles de IAM para aplicaciones que se ejecutan en instancias de Amazon EC2.

Configuración avanzada para su proyecto de AWS SDK for .NET

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

(Antes, este tema se titulaba “Configuración de AWS SDK for .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 usar AWS SDK for .NET con .NET Core más fácilmente, puede utilizar el paquete NuGet AWSSDK.Extensions.NETCore.Setup. Como muchas bibliotecas de .NET Core, agrega métodos de extensión a la interfaz `IConfiguration` para obtener la configuración de AWS de forma fluida.

Uso de AWSSDK.Extensions.NETCore.Setup

Imaginemos que creamos una aplicación Model-View-Controller (MVC) de ASP.NET Core, algo que podemos realizar con la plantilla Aplicación web de ASP.NET Core en Visual Studio o ejecutando `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 objeto `Configuration` para obtener las opciones de AWS, agregue primero el paquete NuGet `AWSSDK.Extensions.NETCore.Setup`. 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. Cuando se implementa una instancia de Amazon EC2 que tiene `EnvironmentName` establecido en Production, este archivo se omite y AWS SDK for .NET utiliza las credenciales y la región de IAM 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 obtener acceso a las opciones de AWS establecidas en el archivo en el código, llame al método de extensión `GetAWSOptions` que se ha 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 agregar el paquete NuGet AWSSDK.S3 al proyecto).

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

El paquete NuGet AWSSDK.Extensions.NETCore.Setup también se integra con un nuevo sistema de inserción 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 paquete NuGet `AWSSDK.Extensions.NETCore.Setup` agrega nuevos métodos de extensión a `IServiceCollection` que se pueden usar para agregar servicios de AWS a la inserción de dependencias. En el siguiente código se muestra cómo agregar las opciones de AWS que se leen de `IConfiguration` para agregar Amazon S3 y DynamoDB a la lista de servicios (asegúrese de agregar los paquetes AWSSDK.S3 y AWSSDK.DynamoDBv2 al 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`
- Puntos de conexión generados por servicios de AWS

Estos parámetros se pueden configurar en el archivo `App.config` o `Web.config` de la aplicación. Aunque también puede configurarlos en la API del AWS SDK for .NET, le recomendamos que utilice el archivo `.config` de la aplicación. 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 for .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. Esta es la opción predeterminada.

`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"/>
```

De forma alternativa, al utilizar la API del AWS SDK for .NET, combine los valores de la enumeración LoggingOptions y defina la propiedad AWSConfigs.Logging:

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

Los cambios en esta configuración solo son efectivos para las instancias del cliente AWS nuevas.

AWSLogMetrics

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">
```

De forma alternativa, para definir el registro de las métricas con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.LogMetrics:

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

Esta opción configura la propiedad `LogMetrics` de forma predeterminada para todos los clientes y configuraciones. Los cambios en esta configuración solo son efectivos para las instancias del cliente AWS nuevas.

AWSRegion

Configura la región de AWS predeterminada para 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"/>
```

De forma alternativa, para definir la región con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.AWSRegion:

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

Para obtener más información sobre cómo crear un cliente de AWS para una región específica, consulte Selección de la región de AWS. Los cambios en esta configuración solo son efectivos para las instancias del cliente AWS nuevas.

AWSResponseLogging

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

`Never`

No registrar nunca respuestas del servicio. Esta es la opción predeterminada.

`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>
```

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

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

De forma alternativa, para definir el registro de servicios con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.ResponseLogging con uno de los valores de la enumeración ResponseLoggingOption:

```
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-"/>
```

De forma alternativa, para definir el prefijo del nombre de tabla con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.DynamoDBContextTableNamePrefix:

```
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"/>
```

De forma alternativa, para definir la versión 4 de Signature con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.S3UseSignatureVersion4 como `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.

AWSEndpointDefinition

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"/>
```

De forma alternativa, puede definir la clave AWSEndpointDefinition en la sección `<appSettings>`:

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

De forma alternativa, para definir el archivo de definición del punto de enlace con la API del AWS SDK for .NET, defina la propiedad AWSConfigs.EndpointDefinition:

```
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 son efectivos para las instancias del cliente AWS nuevas. El archivo endpoint.json está disponible en https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json.

Puntos de conexión generados por servicios de AWS

Algunos servicios de AWS generan sus propios puntos de conexión en lugar de consumir un punto de conexión de región. 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 puntos de conexión de Amazon CloudSearch

El cliente de Amazon CloudSearch se utiliza para acceder al servicio de configuración de Amazon CloudSearch. Debe utilizar el servicio de configuración de Amazon CloudSearch para crear, configurar y administrar los dominios de búsqueda. Para crear un dominio de búsqueda, cree un objeto CreateDomainRequest y proporcione la propiedad `DomainName`. Cree un objeto AmazonCloudSearchClient mediante el objeto de la solicitud. Llame al método CreateDomain. El objeto CreateDomainResponse devuelto por la llamada contiene una propiedad `DomainStatus` que incluye los puntos de enlace `DocService` y `SearchService`. Cree un objeto AmazonCloudSearchDomainConfig y utilícelo para inicializar las instancias `DocService` y `SearchService` de la clase AmazonCloudSearchDomainClient.

```
// 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);
}
```

Ejemplo de puntos de enlace de AWS IoT

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

```
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 los archivos de configuración para el AWS SDK for .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 archivo `App.config` o `Web.config` del proyecto de .NET para especificar la configuración de AWS, como credenciales de AWS, opciones de registro, puntos de conexión de servicio de AWS y regiones de AWS, así como algunos ajustes de servicios de AWS, como Amazon DynamoDB, Amazon EC2 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**
>
> Aunque puede continuar usando el elemento `<appSettings>` en un archivo `App.config` o `Web.config` para especificar la configuración de AWS, recomendamos usar los elementos `<configSections>` y `<aws>`, como se describe posteriormente en este tema. Para obtener más información sobre el elemento `<appSettings>`, consulte los ejemplos del elemento `<appSettings>` en Configuración de la aplicación de AWS SDK for .NET.

> **Note**
>
> Aunque puede continuar usando las siguientes propiedades de la clase AWSConfigs en un archivo de código para especificar la configuración de AWS, las siguientes propiedades están en desuso y es posible que no se admitan en versiones futuras:
>
> - `DynamoDBContextTableNamePrefix`
> - `EC2UseSignatureVersion4`
> - `LoggingOptions`
> - `LogMetrics`
> - `ResponseLoggingOption`
> - `S3UseSignatureVersion4`
>
> En general, recomendamos que, en lugar de usar propiedades de la clase `AWSConfigs` en un archivo de código para especificar la configuración de AWS, debe usar los elementos `<configSections>` y `<aws>` en un archivo `App.config` o `Web.config` para especificar la configuración de AWS, como se describe posteriormente en este tema. Para obtener

más información sobre las propiedades anteriores, consulte los ejemplos de código de `AWSConfigs` en Configuración de la aplicación de AWS SDK for .NET.

Temas

- Declaración de una sección de configuración de AWS
- Elementos permitidos
- Referencia de elementos

Declaración de una sección de configuración de AWS

La configuración de AWS se especifica en un archivo `App.config` o `Web.config` desde el elemento `<aws>`. 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 relleno de código automático (IntelliSense) para el elemento `<aws>` 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

A continuación se muestra una lista de los elementos permitidos en una sección de configuración de AWS. Para cada elemento, se listan sus atributos y elementos principales y secundarios permitidos.

Temas

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 for .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 for .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 for .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 elemento `<aws>` representa el elemento superior en una sección de configuración de AWS. Este elemento puede incluir los siguientes atributos:

`endpointDefinition`

Ruta absoluta a un archivo de configuración personalizado que define las regiones y los puntos de conexión de AWS que se van a usar. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.EndpointDefinition` en el AWS SDK for .NET.

`profileName`

Nombre de perfil de las credenciales de AWS almacenadas que se usarán para realizar llamadas a servicios. Este atributo se asigna a la propiedad `Amazon.AWSConfigs.AWSProfileName` en el AWS SDK for .NET.

`profilesLocation`

Ruta absoluta a la ubicación del archivo de credenciales compartido con otros AWS SDK. 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 for .NET.

`region`

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

dynamoDBContext

El elemento <dynamoDBContext> representa una colección de ajustes específicos del contexto de Amazon DynamoDB. Este elemento puede incluir el atributo tableNamePrefix, que representa el prefijo de nombre de tabla predeterminado que usará 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 for .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 elemento <ec2> representa una colección de ajustes de Amazon EC2. El elemento puede incluir el atributo useSignatureVersion4, que especifica si la versión 4 de Signature se usará para todas las solicitudes (true) o no (false, el valor predeterminado). Este atributo se asigna a la propiedad `Amazon.Util.EC2Config.UseSignatureVersion4` a partir de la propiedad `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` del AWS SDK for .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 for .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 for .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 for .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 for .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 se deben registrar (se asigna a la propiedad `LogTo` a partir de la propiedad `Amazon.AWSConfigs.LoggingConfig.LogTo` en el AWS SDK for .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 elemento <map> representa un solo elemento de una colección de mapeos de tipo a tabla de tipos de .NET a tablas de DynamoDB (se asigna a una instancia de la clase TypeMapping a partir de la propiedad Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings en el AWS SDK for .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 for .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 for .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>
```

property

El elemento <property> representa una propiedad DynamoDB. (Este elemento se asigna a una instancia de la clase Amazon.Util.PropertyConfig a partir del método AddProperty en el AWS SDK for .NET). Para obtener más información, consulte las secciones sobre las mejoras realizadas 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 for .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 for .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 for .NET.

`name`

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

password

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 for .NET.

port

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 for .NET.

username

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 for .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. El elemento puede incluir el atributo useSignatureVersion4, que especifica si la versión 4 de Signature se usará para todas las solicitudes (true) o no (false, el valor predeterminado). Este atributo se asigna a la propiedad `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` en el AWS SDK for .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 autenticarse en la Guía de referencia de herramientas y AWS SDK.
> Para conocer las prácticas recomendadas de seguridad, use AWS IAM Identity Center, como se describe en Configuración de la autenticación de SDK.

Advertencias y directrices importantes para las credenciales

Advertencias para las credenciales

- NO use las credenciales raíz de la cuenta para obtener acceso a los recursos de AWS. Estas credenciales proporcionan acceso ilimitado a la cuenta y son difíciles de revocar.

- NO incluya claves de acceso literales ni información sobre credenciales en los 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 las credenciales almacenadas en el archivo compartido `credentials` de AWS se almacenan en texto no cifrado.

Guía adicional para administrar las credenciales de forma segura

Para obtener información general sobre cómo administrar de forma segura las credenciales de AWS, consulte credenciales de seguridad de AWS en Referencia general de AWS y Prácticas recomendadas de seguridad y casos de uso 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 credenciales de AWS 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. Los otros AWS SDK pueden usar este archivo. También lo pueden utilizar la AWS CLI, las AWS Tools for Windows PowerShell y los kits de herramientas de AWS para Visual Studio, JetBrains y VS Code.

> **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 autenticarse en la Guía de referencia de herramientas y AWS SDK.
> Para conocer las prácticas recomendadas de seguridad, use AWS IAM Identity Center, como se describe en Configuración de la autenticación de SDK.

Información general

El archivo de credenciales de AWS compartido se encuentra de forma predeterminada en el directorio `.aws` del directorio principal y se denomina `credentials`; es decir, `~/.aws/credentials` (Linux o macOS) o `%USERPROFILE%\.aws\credentials` (Windows). Para obtener información sobre otras ubicaciones, consulte Ubicación de archivos compartidos en la Guía de referencia de herramientas y AWS SDK. Consulte también Acceso a las credenciales y perfiles en una aplicación.

El archivo de credenciales de AWS compartido es un archivo de texto sin formato y tiene un formato determinado. Para obtener información sobre el formato de los archivos de credenciales de AWS, consulte Formato de los archivos de credenciales en la Guía de referencia de herramientas y AWS SDK.

Los perfiles del archivo de credenciales de AWS compartido se pueden administrar de varias formas.

- Con un editor de texto para crear y actualizar el archivo de credenciales de AWS compartido
- Con el espacio de nombres Amazon.Runtime.CredentialManagement de la API de AWS SDK for .NET, como explicaremos más adelante en este tema
- Con los comandos y procedimientos de AWS Tools for PowerShell y los kits de herramientas de AWS para Visual Studio, JetBrains y VS Code
- 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 del archivo de credenciales de AWS compartido. 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 credenciales de AWS compartido casi siempre tendrá un perfil denominado default. Aquí es donde AWS SDK for .NET busca las credenciales cuando 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 credenciales de AWS compartido mediante programación. Se utilizan las siguientes clases del espacio de nombres Amazon.Runtime.CredentialManagement: CredentialProfileOptions, CredentialProfile y SharedCredentialsFile.

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

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

```
...

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

> **Warning**
>
> 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. Se utilizan las siguientes clases del espacio de nombres Amazon.Runtime.CredentialManagement: CredentialProfile y SharedCredentialsFile. También se usa la clase RegionEndpoint 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**
>
> La región de AWS también se puede establecer en otras ubicaciones y mediante otros métodos. Para obtener más información, consulte Configuración de la región de AWS.

Uso de SDK Store (solo Windows)

(Asegúrese de revisar las advertencias y directrices importantes).

En Windows, SDK Store es otro lugar donde crear perfiles y almacenar las credenciales cifradas de la aplicación AWS SDK for .NET. 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, consulte Otras formas de autenticarse en la Guía de referencia de herramientas y AWS SDK.
> Para conocer las prácticas recomendadas de seguridad, use AWS IAM Identity Center, como se describe en Configuración de la autenticación de SDK.

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
- Con el espacio de nombres Amazon.Runtime.CredentialManagement de la API de AWS SDK for .NET, como explicaremos 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. Se utilizan las siguientes clases del espacio de nombres Amazon.Runtime.CredentialManagement: CredentialProfileOptions, CredentialProfile y NetSDKCredentialsFile.

```
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. Se utilizan las siguientes clases del espacio de nombres Amazon.Runtime.CredentialManagement: CredentialProfile y NetSDKCredentialsFile. También se usa la clase RegionEndpoint 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**
>
> La región de AWS también se puede establecer en otras ubicaciones y mediante otros métodos. Para obtener más información, consulte Configuración de la región de AWS.

Características de AWS SDK for .NET

En esta sección se proporciona información sobre las características de AWS SDK for .NET que le puede interesar tener en cuenta al crear sus aplicaciones.

Antes de nada, asegúrese de haber configurado el proyecto.

Para obtener información sobre el desarrollo de software para servicios de AWS específicos junto con ejemplos de código, consulte Trabaje con AWS los servicios. Para ver otros ejemplos de código, consulte AWS SDK for .NET ejemplos de código.

Temas

API asincrónicas de AWS para .NET

AWS SDK for .NET utiliza el patrón asincrónico basado en tareas (TAP) en su implementación asincrónica. Para obtener más información sobre el TAP, consulte Patrón asincrónico basado en tareas (TAP) en docs.microsoft.com.

En este tema se ofrece información general sobre cómo utilizar el TAP en las llamadas a clientes de servicio de AWS.

Los métodos asincrónicos de la API de AWS SDK for .NET son operaciones basadas en las clases `Task` o `Task<TResult>`. Consulte docs.microsoft.com para obtener información sobre estas clases: Clase Task, Clase Task<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.

Encontrará ejemplos completos de llamadas asincrónicas a clientes de servicio de AWS en la sección Recorrido rápido (Aplicación multiplataforma sencilla y Aplicación sencilla basada en Windows) y en Ejemplos de código con orientaciones.

Reintentos y tiempos de espera

AWS SDK for .NET permite configurar el número de reintentos y los valores de tiempo de espera de las solicitudes HTTP relativas a servicios de AWS. 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 AWS SDK for .NET y la aplicación cuando la conectividad de red se degrade o no se pueda tener acceso a un servicio de AWS? ¿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

Información general

AWS SDK for .NET reintentará las solicitudes con errores producidos por la limitación controlada del lado del servidor o por conexiones perdidas. 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. ClientConfigclase de la referencia de la APIAWS SDK for .NET:

- `RetryMode`especifica uno de los tres modos de reintento, que se definen en Amazon.Runtime. RequestRetryModeenumeración.

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

- `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 AWS compartido.

Las descripciones detalladas de estas propiedades se encuentran en el resumen Amazon.Runtime.ClientConfigclase de la AWS SDK for .NETAPI Reference. Cada valor de `RetryMode` se corresponde de forma predeterminada con un valor concreto de `MaxErrorRetry`, como se muestra en la siguiente tabla.

RetryMode	Corresponding MaxErrorRetry (Amazon DynamoDB)	Corresponding MaxErrorRetry (all others)
Legacy	10	4
Standard	10	2
Adaptive (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 AmazonDynamoDBConfig o AmazonSQSConfig 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 for .NET permite configurar los valores de tiempo de espera de la solicitud o de tiempo de espera de lectura/escritura del socket en el nivel de cliente del servicio. Estos valores se especifican en las `ReadWriteTimeout` propiedades del `Timeout` resumen Amazon.Runtime. ClientConfigclase. Estos valores se transmiten como `ReadWriteTimeout` propiedades de `Timeout` los HttpWebRequestobjetos 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 for .NET que se dirige a la biblioteca de clases portátil (PCL) utiliza la HttpClientclase 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 for .NET AmazonGlacierClient
- Las versiones AWS SDK for .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

Ejemplo

En 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 tiempo de espera de lectura/escritura de 10 segundos (si procede). 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 for .NET that target .NET Core.
    ReadWriteTimeout = TimeSpan.FromSeconds(10),
    RetryMode = RequestRetryMode.Standard,
    MaxErrorRetry = 3
  });
```

Paginadores

Algunos AWS servicios recopilan y almacenan una gran cantidad de datos, que puede recuperar mediante las llamadas a la API del AWS SDK for .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 for .NET.

Por ejemplo, si examina la definición de la AmazonCloudWatchLogsClientclase, verá una `Paginators` 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 DescribeLogGroupsResponseobjeto 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 for .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).

  > **Warning**
  >
  > A partir del 15 de agosto de 2024, AWS SDK for .NET dejará de ser compatible con .NET Framework 3.5 y cambiará 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 3.5 y 4.5 de .NET Framework. AWS SDK for .NET

  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 asincrónica.

Herramientas adicionales

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

Herramienta de implementación de AWS

Una vez desarrollada la aplicación .NET Core nativa en la nube en un equipo de desarrollo, puede usar la Herramienta de implementación de AWS de la CLI de .NET para implementar la aplicación más fácilmente en AWS.

Para obtener más información, consulte Implementación de aplicaciones en AWS.

AWS Marco de procesamiento de mensajes para .NET

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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.

Autenticación y autorización avanzadas con AWS SDK for .NET

En los temas de esta sección se proporciona información sobre técnicas avanzadas de autenticación y autorización en la aplicación AWS SDK for .NET.

Temas

Inicio de sesión único con AWS SDK for .NET

AWS IAM Identity Center es un servicio de inicio de sesión único (SSO) basado en la nube que facilita la administración centralizada de acceso de SSO a todas las Cuentas de AWS y aplicaciones en la nube. 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

En términos generales, los SDK interactúan con IAM Identity Center de forma similar al siguiente patrón:

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 archivo `config` de AWS compartido del equipo del usuario se actualiza con la información de SSO.
3. El usuario inicia sesión a través de IAM Identity Center y recibe unas credenciales de duración limitada correspondientes a los permisos de AWS Identity and Access Management (IAM) que se han configurado para él. Este inicio de sesión se puede realizar a través de una herramienta que no pertenezca a SDK, como AWS CLI, o bien mediante programación usando 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 Configuración de la autenticación de SDK. Si no está familiarizado con el uso de SSO en AWS, quizás le interese consultar primero ese tema para obtener información fundamental y, después, pasar a los siguientes tutoriales para ver SSO en acción:

- Tutorial: solo aplicación .NET
- Tutorial: AWS CLI y aplicación .NET

Este tema contiene las siguientes secciones:

Requisitos previos

Antes de utilizar IAM Identity Center, debe realizar algunas tareas, como seleccionar un origen de identidad y configurar las Cuentas de AWS y las aplicaciones pertinentes. 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 que IAM Identity Center esté configurado en la Cuenta de AWS correspondiente, hay que agregar un perfil con nombre para SSO al archivo `config` de AWS compartido del usuario. 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 acceso de AWS. 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`

Región de 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 los puntos de conexiones regionales de Regiones de AWS, consulte Regiones y puntos de conexión en la Referencia general de Amazon Web Services.

`sso_account_id`

ID de una Cuenta de AWS que se ha agregado a través del servicio de AWS Organizations. Para ver la lista de cuentas disponibles, vaya a la consola del Centro de identidades de IAM 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 SSO, primero el usuario debe generar un token temporal y, a continuación, usarlo para acceder a los recursos y las aplicaciones de AWS relevantes. 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
- Generar un token con AWS CLI y, a continuación, usarlo 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 paquetes NuGet para que la resolución de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

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

Aparte de con AWS CLI, también se puede 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 paquetes NuGet para que la resolución de SSO funcione:

- `AWSSDK.SSO`
- `AWSSDK.SSOOIDC`

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

```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
```

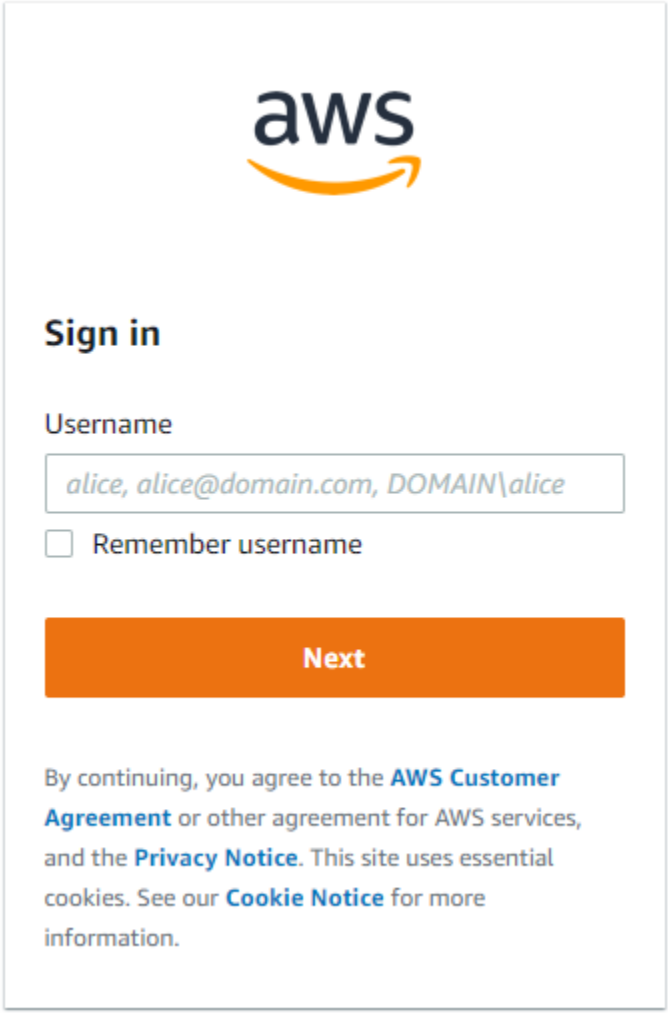
```
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    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
        });
    };

    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:



> **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 describe cómo generar un token de SSO temporal mediante AWS CLI y cómo usarlo en una aplicación. Para ver un tutorial completo de este proceso, consulte Tutorial de SSO mediante AWS CLI y aplicaciones .NET.

Generación de un token de SSO mediante AWS CLI

Además de generar un token de SSO temporal mediante programación, también se puede generar con AWS CLI. 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**
>
> La aplicación debe hacer referencia a los siguientes paquetes NuGet para que la resolución de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 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 servicio de AWS sin parámetros, de forma similar a "`var client = new AmazonS3Client();`".

Tutorial de SSO mediante AWS CLI y aplicaciones .NET

Recursos adicionales

Para obtener más ayuda, consulte los recursos siguientes:

- ¿Qué es IAM Identity Center?
- Configurar AWS CLI para usar IAM Identity Center
- Uso de credenciales de IAM Identity Center en el AWS Toolkit for Visual Studio

Tutoriales

Temas

- Tutorial de SSO utilizando únicamente aplicaciones .NET
- Tutorial de SSO mediante AWS CLI y aplicaciones .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 for .NET. Para obtener información completa sobre cómo usar IAM Identity Center con AWS SDK for .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 ayudan a configurar servicios como AWS Organizations e IAM Identity Center. 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 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.

Configuración de AWS

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

Para realizar esta configuración, inicie sesión primero en la Cuenta de AWS de prueba 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 IAM

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 Cuenta de AWS de prueba a la organización como la 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 banner de la AWS Management Console, busque la Región de AWS que se configuró al habilitar el SSO. Es el menú desplegable a la izquierda del ID de Cuenta de AWS. 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, haga clic en 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:

```
{
    "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. Vaya a la página Cuentas de AWS y seleccione la cuenta de AWS que agregó 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

Incluya los paquetes NuGet `AWSSDK.SSO` y `AWSSDK.SSOOIDC`, 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.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.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($"\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");

            var ssoCredentials = credentials as SSOAWSCredentials;

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

```
            return ssoCredentials;
        }

    }

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

Enumerar usuarios de IAM

Incluya los paquetes NuGet `AWSSDK.SSO` y `AWSSDK.SSOOIDC`, además de `AWSSDK.IdentityManagement` y `AWSSDK.SecurityToken`.

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


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

namespace SSOExample.IAM.Programmatic_login
{
    class Program
    {
        // Requirements:
```

```
        // - An SSO profile in the SSO user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

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

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }

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

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
login.
```

```
                //  This method is only invoked if the session doesn't already have a
 valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
 or Linux. If not,
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };

            return ssoCredentials;
        }

    }

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

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 llevan a cabo tareas de inicio de sesión único proporcionando un método de devolución de llamada en la propiedad Opciones 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 archivo `config` de AWS 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. Si es necesario, cree un archivo `config` de AWS compartido en esa carpeta 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 recursos de AWS o recursos del entorno de desarrollo, como archivos y carpetas.

Tutorial de SSO mediante AWS CLI y aplicaciones .NET

En este tutorial se explica cómo habilitar SSO para una aplicación .NET básica y un usuario de SSO de prueba. En él se utiliza 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 en AWS SDK for .NET. Para obtener información completa sobre cómo usar IAM Identity Center con AWS SDK for .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 ayudan a configurar servicios como AWS Organizations e IAM Identity Center. 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 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 tener instalada la versión 2 de AWS CLI en el equipo del usuario de SSO. Para comprobarlo, ejecute `aws --version` en una línea de comandos o en un terminal.

Configuración de AWS

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

Para realizar esta configuración, inicie sesión primero en la Cuenta de AWS de prueba 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 IAM

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 Cuenta de AWS de prueba a la organización como la 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 banner de la AWS Management Console, busque la Región de AWS que se configuró al habilitar el SSO. Es el menú desplegable a la izquierda del ID de Cuenta de AWS. 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, haga clic en 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:

```
{
    "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. Vaya a la página Cuentas de AWS y seleccione la cuenta de AWS que agregó 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

Incluya los paquetes NuGet `AWSSDK.SSO` y `AWSSDK.SSOOIDC`, además de `AWSSDK.S3` y `AWSSDK.SecurityToken`.

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

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
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 los paquetes NuGet `AWSSDK.SSO` y `AWSSDK.SSOOIDC`, además de `AWSSDK.IdentityManagement` y `AWSSDK.SecurityToken`.

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

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
 AWSSDK.SSOOIDC
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 archivo `config` de AWS 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. Si es necesario, cree un archivo `config` de AWS compartido en esa carpeta 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 comando AWS CLI:

   ```
   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 recursos de AWS o recursos del entorno de desarrollo, como archivos y carpetas.

Implementación de 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. Esto se puede hacer con la AWS Management Console o usando servicios como AWS CloudFormation o AWS Cloud Development Kit (AWS CDK). También se pueden usar herramientas de AWS que se han creado con el propósito de realizar implementaciones. Con estas herramientas se puede hacer lo siguiente.

Implementación desde la CLI de .NET

Para implementar sus aplicaciones en AWS, puede usar las siguientes herramientas de AWS de la CLI de .NET:

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

Implementación desde kits de herramientas de IDE

Se pueden utilizar kits de herramientas de AWS para implementar sus aplicaciones directamente desde el IDE de su elección:

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > La característica “Publicar en AWS” del kit de herramientas tiene la misma funcionalidad que la Herramienta de implementación de AWS para la CLI de .NET. Para obtener más información, consulte Publicación en AWS en la Guía del usuario de AWS Toolkit for Visual Studio.

- AWS Toolkit for JetBrains

  Consulte Uso de aplicaciones sin servidor de AWS y Uso del AWS App Runner.

- Kit de herramientas de AWS para VS Code

Consulte Uso de aplicaciones sin servidor y Uso de AWS App Runner.

- AWS Toolkit for 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
- Proyectos de AWS Lambda

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 recurso informático para la aplicación: obtenga recomendaciones de recurso informático y descubra qué recurso informático de AWS 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 implementación de AWS CDK generado e implementa la aplicación en el recurso informático de AWS escogido.
- Implementaciones reproducibles y compartibles: puede generar y modificar proyectos de implementación de AWS CDK para adaptarlos a sus circunstancias específicas. También puede controlar las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.
- Ayuda para conocer AWS CDK para .NET: la herramienta le ayuda a conocer poco a poco las herramientas de AWS subyacentes en las que se basa, como AWS CDK.

La Herramienta de implementación de AWS permite implementar aplicaciones ASP.NET Core en los siguientes servicios de AWS:

- Amazon ECS Service con AWS Fargate: admite implementaciones de aplicaciones web en Amazon Elastic Container Service (Amazon ECS) con una potencia de computación administrada por un motor de computación sin servidor de AWS Fargate.
- AWS App Runner: admite implementaciones en un servicio completamente administrado que facilita a los desarrolladores la implementación de API y aplicaciones web en contenedores a escala. No se requiere experiencia previa en infraestructura.
- AWS Elastic Beanstalk: admite implementaciones en un servicio que facilita a los desarrolladores la implementación de API y aplicaciones web en un entorno completamente administrado 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 recurso informático para la aplicación: obtenga recomendaciones de recurso informático y descubra qué recurso informático de AWS 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 implementación de AWS CDK generado e implementa la aplicación en el recurso informático de AWS escogido.
- Implementaciones reproducibles y compartibles: puede generar y modificar proyectos de implementación de AWS CDK para adaptarlos a sus circunstancias específicas. También

puede controlar las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.
- Ayuda para conocer AWS CDK para .NET: la herramienta le ayuda a conocer poco a poco las herramientas de AWS subyacentes en las que se basa, como AWS CDK.

La Herramienta de implementación de AWS permite implementar aplicaciones de la consola de .NET en los siguientes servicios de AWS:

- Amazon ECS Service con AWS Fargate: admite implementaciones de aplicaciones .NET como un servicio (por ejemplo, un procesador en segundo plano) en Amazon Elastic Container Service (Amazon ECS) con una potencia de computación administrada por un motor de computación sin servidor de AWS Fargate.
- 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, carga la aplicación Blazor en 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 implementación de AWS CDK generado e implementa la aplicación en el recurso informático de AWS escogido.
- Implementaciones reproducibles y compartibles: puede generar y modificar proyectos de implementación de AWS CDK para adaptarlos a sus circunstancias específicas. También puede controlar las versiones de sus proyectos y compartirlos con su equipo para realizar implementaciones reproducibles.

- Ayuda para conocer AWS CDK para .NET: la herramienta le ayuda a conocer poco a poco las herramientas de AWS subyacentes en las que se basa, como 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.

Proyectos de AWS Lambda

AWS Lambda es un servicio informático que 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 desarrolladores de AWS Lambda.

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

Temas

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 instalarla.
- 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 siguientes instrucciones se da por hecho que ha creado un proyecto de .NET de AWS Lambda. 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. Cuando se le pida, escriba la región de AWS (la región en la que se implementará la función de 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 Utilización de AWS Lambda con otros servicios.

Lambda monitoriza automáticamente las funciones de Lambda en su nombre 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.

Migración del proyecto de AWS SDK for .NET

En esta sección se proporciona información sobre las tareas de migración que pueden encajar en su caso, así como instrucciones sobre cómo realizarlas.

Temas

Qué hay de nuevo en el AWS SDK for .NET

Consulte la página del producto en https://aws.amazon.com/sdk-for-net/para obtener información de alto nivel sobre los nuevos desarrollos relacionados con el AWS SDK for .NET.

Estas son las novedades de AWS SDK for .NET.

16 de agosto de 2024: versión preliminar de la versión 4

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

La AWS SDK for .NET versión 4 es un cambio evolutivo que modernizará la SDK empresa, resolverá las deficiencias técnicas y abordará las opiniones de los clientes que requieren cambios importantes. La versión 4 se ha publicado como una primera vista previa. Para obtener más información sobre esta versión preliminar y probarla, consulte la entrada del blog titulada Preview 1 of AWS SDK for .NET V4 y el número V4 Development Tracker en GitHub.

28 de marzo de 2024: versión preliminar del marco de procesamiento de AWS mensajes para. NET

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

El marco de procesamiento de AWS mensajes para. NETes un marco AWS nativo que simplifica el desarrollo de. NETaplicaciones de procesamiento de mensajes que utilizan AWS servicios como Amazon Simple Queue Service ()SQS, Amazon Simple Notification Service (SNS) y Amazon. EventBridge

23 de febrero de 2024: se agregó soporte para. NET8

Support for. NETSe agregaron 8 a AWS SDK for .NET. Utilice los NuGet paquetes o ensamblajes más recientes compatibles. NET8 y versiones posteriores. Puede encontrar información adicional sobre este soporte, incluido el soporte para Lambda, en la entrada del blog. NET8 Support activado AWS.

18 de febrero de 2024: Próximos cambios en. NETSoporte marco

A partir del 15 de agosto de 2024, AWS SDK for .NET finalizará el soporte para. NETFramework 3.5 y cambiará el mínimo. NETVersión Framework a la 4.7.2. Para obtener más información, consulte la entrada del blog Se avecinan cambios importantes. NETObjetivos de los marcos 3.5 y 4.5 del AWS SDK for .NET.

17 de julio de 2021: Se ha publicado el marco de AWS Lambda anotaciones para su disponibilidad general

El marco de AWS Lambda anotaciones hace que la experiencia de escribir funciones Lambda en C# parezca más natural. NETdesarrolladores utilizando la tecnología de generación de fuentes de C#. Ya está disponible con carácter general.

15-07-2023: Se ha publicado una versión preliminar del proveedor de caché distribuida de DynamoDB

> Esta es una documentación previa a la versión final de una característica en versión preliminar. Está sujeta a cambios.

La biblioteca Distributed Cache Provider permite utilizar Amazon DynamoDB como almacenamiento para. ASP NETEl marco de caché distribuida de Core. Para obtener más información, consulte la entrada del blog Introducing theAWS. NETProveedor de caché distribuida para DynamoDB (versión preliminar) y el repositorio. GitHub

13 de julio de 2022: Se ha lanzado la herramienta de despliegue AWS

Se ha lanzado la herramienta de AWS despliegue. Esta herramienta es una herramienta interactiva para. NETCLIy lo AWS Toolkit for Visual Studio que ayuda a implementar. NETaplicaciones con AWS conocimientos mínimos y con el menor número de clics o comandos. Para obtener más información, consulte Implementación de aplicaciones en AWS.

24 de agosto de 2020: Se ha publicado la versión 3.5 del SDK

- Estandarizó el. NETexperiencia mediante la transición del soporte para todas las variaciones no relacionadas con Framework delSDK. NETEstándar 2.0. Para obtener más información, consulte Migración a la versión 3.5.
- Se han añadido paginadores a muchos clientes de servicios, lo que facilita la paginación de API los resultados. Para obtener más información, consulte Paginadores.

Plataformas compatibles con el AWS SDK for .NET

AWS SDK for .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 for .NET Admite 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 for .NET cumplen con el estándar 2.0 de.NET. AWS SDK for .NET proporciona solo métodos asincrónicos para las aplicaciones escritas con .NET Standard.

.NET Framework 4.5

> **Warning**
>
> A partir del 15 de agosto de 2024, AWS SDK for .NET dejarán de ser compatibles con .NET Framework 3.5 y cambiarán 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 3.5 y 4.5 de .NET Framework. AWS SDK for .NET

Esta versión de AWS SDK for .NET está compilada con .NET Framework 4.5 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

> **Warning**
>
> A partir del 15 de agosto de 2024, AWS SDK for .NET dejarán de ser compatibles con .NET Framework 3.5 y cambiarán 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 3.5 y 4.5 de .NET Framework. AWS SDK for .NET

Esta versión de AWS SDK for .NET está compilada con .NET Framework 3.5 y se ejecuta en entornos de ejecución .NET 2.0 o .NET 4.0. AWS los clientes de servicio admiten patrones de llamadas síncronos y asíncronos y utilizan el patrón anterior de inicio y fin.

> **Note**
>
> No AWS SDK for .NET cumple con el Estándar Federal de Procesamiento de Información (FIPS) cuando se utiliza en aplicaciones creadas con la versión 2.0 del CLR. Para obtener más información sobre cómo sustituir una implementación compatible con FIPS en ese entorno, consulte el blog de Microsoft y la clase HMACSHA256 (HMACSHA256CNG) del equipo de seguridad de CLR CryptoConfigen Security.Cryptography.dll.

Biblioteca de clases portátil y Xamarin

También contiene una implementación de biblioteca de clases portátil. AWS SDK for .NET 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.

Más información

Migración a la versión 3.5 de AWS SDK for .NET

Migración a la versión 3 de la AWS SDK for .NET

En este tema se describen los cambios en la versión 3 del AWS SDK for .NET y cómo migrar su código a esta versión del SDK.

Acerca de las versiones del AWS SDK for .NET

El AWS SDK for .NET, lanzado originalmente en noviembre de 2009, se diseñó para .NET Framework 2.0. Desde entonces, .NET se ha mejorado con .NET Framework 4.0 y .NET Framework 4.5, y se han añadido nuevas plataformas de destino: WinRT y Windows Phone.

AWS SDK for .NET versión 2 se actualizó para beneficiarse de las nuevas características de la plataforma .NET y centrarse en WinRT y Windows Phone.

AWS SDK for .NET versión 3 se actualizó para que los ensamblados sean modulares.

Rediseño de la arquitectura del SDK

Toda la versión 3 del AWS SDK for .NET se ha rediseñado para que sea modular. Ahora cada servicio se implementa en su propio ensamblado, en lugar de en un ensamblado global. Ya

no es necesario que añada todo el AWS SDK for .NET en su aplicación. Ahora puede agregar ensamblados solo para los servicios de AWS que su aplicación utiliza.

Cambios bruscos

En las siguientes secciones se describen cambios de la versión 3 del AWS SDK for .NET.

AWSClientFactory eliminada

Se ha eliminado la clase `Amazon.AWSClientFactory`. Ahora, para crear un cliente de servicio debe utilizar el constructor del cliente de servicio. Por ejemplo, para crear un `AmazonEC2Client`:

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

Amazon.Runtime.AssumeRoleAWSCredentials eliminada

Se ha eliminado la clase `Amazon.Runtime.AssumeRoleAWSCredentials` porque estaba en un espacio de nombres central pero era dependiente de AWS Security Token Service y porque ya estaba obsoleta en el SDK desde hacía tiempo. En su lugar, utilice la clase `Amazon.SecurityToken.AssumeRoleAWSCredentials`.

Método SetACL eliminado de S3Link

La clase `S3Link` forma parte del paquete `Amazon.DynamoDBv2`, y se utiliza para almacenar objetos en Amazon S3 que son referencias en un elemento de DynamoDB. Se trata de una característica útil, pero no queríamos crear una dependencia de compilación en el paquete `Amazon.S3` para DynamoDB. En consecuencia, hemos simplificado los métodos `Amazon.S3` expuestos de la clase `S3Link` reemplazando el método `SetACL` por el método `MakeS3ObjectPublic`. Para obtener más control sobre la lista de control de acceso (ACL) en el objeto, utilice el paquete `Amazon.S3` directamente.

Eliminación de clases de resultado obsoletas

Para la mayoría de servicios en el AWS SDK for .NET, las operaciones devuelven un objeto de respuesta que contiene metadatos para la operación, como el ID de la solicitud y un objeto de resultado. Disponer de una respuesta separada y de una clase de resultado era redundante y generaba más trabajo para los desarrolladores. En la versión 2 del AWS SDK for .NET, incluimos toda la información de la clase de resultado en la clase de respuesta. También marcamos las clases de resultado como obsoletas para evitar que se utilizaran. En la versión 3 del AWS SDK for .NET, eliminamos estas clases de resultado obsoletas para reducir el tamaño del SDK.

Cambios en la sección de AWS Config

Se puede realizar una configuración avanzada del AWS SDK for .NET mediante el archivo `App.config` o `Web.config`. Para ello puede utilizar una sección de configuración `<aws>`, como la que se muestra a continuación, que hace referencia al nombre del ensamblado del SDK.

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

En la versión 3 del AWS SDK for .NET, el ensamblado `AWSSDK` ya no existe. Hemos colocado el código común en el ensamblado `AWSSDK.Core`. En consecuencia, deberá modificar las referencias al ensamblado `AWSSDK` en su archivo `App.config` o `Web.config` por el ensamblado `AWSSDK.Core`, de la siguiente manera.

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

También puede manipular las opciones de configuración con la clase `Amazon.AWSConfigs`. En la versión 3 de AWS SDK for .NET, hemos trasladado las opciones de configuración de DynamoDB de la clase `Amazon.AWSConfigs` a la clase `Amazon.AWSConfigsDynamoDB`.

Migración a la versión 3.5 de AWS SDK for .NET

La versión 3.5 de AWS SDK for .NET estandariza aún más la experiencia con .NET mediante la transición del soporte para todas las variaciones del SDK que no sean de Framework a .NET Standard 2.0. En función del entorno y la base de código, para aprovechar las características de la versión 3.5, es posible que necesite realizar determinados trabajos de migración.

En este tema se describen los cambios en la versión 3.5 y el posible trabajo que debe realizar para migrar el entorno o el código desde la versión 3.

¿Qué ha cambiado en la versión 3.5?

A continuación se describe lo que ha cambiado o no en la versión 3.5 de AWS SDK for .NET.

.NET Framework y .NET Core

La compatibilidad para .NET Framework y .NET Core no ha cambiado.

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.

Unity

Las aplicaciones de Unity deben dirigirse a perfiles de .NET Standard 2.0 o .NET 4.x con Unity 2018.1 o una versión posterior. Para obtener más información, consulte Compatibilidad con perfiles de .NET. Además, si utiliza IL2CPP para compilar, debe deshabilitar la extracción de código agregando un archivo link.xml, como se describe en Referencing the AWS SDK for .NET Standard 2.0 from Unity, Xamarin, or UWP. Después de portar el código a una de las bases de código recomendadas, la aplicación Unity puede acceder a todos los servicios ofrecidos por el SDK.

Dado que Unity es compatible con .NET Standard 2.0, el paquete AWSSDK.Core del SDK versión 3.5 ya no tiene código específico de Unity, incluidas algunas funcionalidades de nivel superior. Para proporcionar una mejor transición, todo el código de Unity heredado está disponible como referencia en el repositorio de GitHub aws/aws-sdk-unity-net. Si ve que falta alguna funcionalidad que afecta al uso de AWS con Unity, puede presentar una solicitud de característica en https://github.com/aws/dotnet/issues.

Consulte también Consideraciones especiales sobre la compatibilidad con Unity.

Plataforma universal de Windows

Dirija su aplicación UWP a la versión 16299 o posterior (actualización de Fall Creators, versión 1709, publicada en octubre de 2017).

Windows Phone y Silverlight

La versión 3.5 de AWS SDK for .NET no admite estas plataformas porque Microsoft ya no las está desarrollando activamente. Para obtener más información, consulte lo siguiente:

- Fin del soporte para Windows 10 Mobile
- Fin del soporte de Silverlight

Bibliotecas de clases portátiles heredadas (PCL basadas en perfiles)

Considere redirigir su biblioteca a .NET Standard. Para obtener más información, consulte Comparación con bibliotecas de clases portátiles de Microsoft.

Amazon Cognito Sync Manager y Amazon Mobile Analytics Manager

Las abstracciones generales que facilitan el uso de Amazon Cognito Sync y Amazon Mobile Analytics se han eliminado de la versión 3.5 de AWS SDK for .NET. AWS AppSync es el reemplazo de Amazon Cognito Sync que se prefiere. Amazon Pinpoint es el reemplazo de Amazon Mobile Analytics que se prefiere.

Si el código se ve afectado por la falta de código de biblioteca de nivel superior para AWS AppSync y Amazon Pinpoint, puede registrar su interés en uno (o ambos) de los siguientes problemas de GitHub: https://github.com/aws/dotnet/issues/20 y https://github.com/aws/dotnet/issues/19. Las bibliotecas de Amazon Cognito Sync Manager y Amazon Mobile Analytics Manager se pueden obtener también en los siguientes repositorios de GitHub: aws/amazon-cognito-sync-manager-net y aws/aws-mobile-analytics-manager-net.

Migración de código sincrónico

La versión 3.5 de AWS SDK for .NET es compatible con .NET Framework y .NET Standard (a través de versiones de .NET Core como .NET Core 3.1, .NET 5, etc.). Las variantes de SDK que cumplen con .NET Standard solo proporcionan métodos asincrónicos, por lo que si desea utilizar .NET Standard, debe cambiar el código sincrónico para que se ejecute de forma asincrónica.

Los siguientes fragmentos de código muestran cómo puede cambiar el código síncrono a código asíncrono. El código de estos fragmentos de código se utiliza para mostrar el número de buckets de Amazon S3.

El código original llama a ListBuckets.

```
private static ListBucketsResponse MyListBuckets()
{
  var s3Client = new AmazonS3Client();
  var response = s3Client.ListBuckets();
  return response;
}

// From the calling function
ListBucketsResponse response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
```

Para usar la versión 3.5 del SDK, llame a ListBucketsAsync en su lugar.

```
private static async Task<ListBucketsResponse> MyListBuckets()
{
  var s3Client = new AmazonS3Client();
  var response = await s3Client.ListBucketsAsync();
  return response;
}


// From an **asynchronous** calling function
ListBucketsResponse response = await MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");

// OR From a **synchronous** calling function
Task<ListBucketsResponse> response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
```

Migración a la versión 3.7 de AWS SDK for .NET

A partir de la versión 3.7, AWS SDK for .NET deja de ser compatible con .NET Standard 1.3.

Para obtener información sobre cómo migrar desde .NET Standard 1.3, consulte Migración desde .NET Standard 1.3.

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, AWS deja de proporcionar soporte para .NET Standard 1.3 en AWS SDK for .NET el 31 de diciembre de 2020.

AWS sigue ofreciendo actualizaciones de servicio y correcciones de seguridad en AWS SDK for .NET con .NET Standard 1.3 como destino hasta el 1 de octubre de 2020. Después de esta fecha, el destino .NET Standard 1.3 ha entrado en modo de mantenimiento, lo que significa que no se han lanzado nuevas actualizaciones; AWS solo ha aplicado correcciones de errores críticos y revisiones de seguridad.

El 31 de diciembre de 2020, el soporte para .NET Standard 1.3 en AWS SDK for .NET llega al fin de su vida útil. Después de esta fecha, no se han aplicado correcciones de errores o revisiones de seguridad. Los artefactos creados con ese destino siguen estando disponibles para su descarga en 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 for .NET.

Si tiene preguntas o dudas, póngase en contacto con AWS Support.

Trabaje con AWS los servicios del AWS SDK for .NET

Las siguientes secciones contienen ejemplos, tutoriales, tareas y guías que muestran cómo utilizarlos AWS SDK for .NET para trabajar con AWS los servicios. Estos ejemplos y tutoriales se basan en una API que el AWS SDK for .NET proporciona. Para ver qué clases y métodos están disponibles en la API, consulte la Referencia de API de AWS SDK for .NET.

Si es la primera vez que lo conoces AWS SDK for .NET, quizás quieras consultar primero el Recorrido rápido 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 comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Temas

- Ejemplos de código con orientación para la AWS SDK for .NET
- Uso de AWS Lambda como servicio de computación
- Bibliotecas y marcos de alto nivel para AWS SDK for .NET
- Programación de AWS OpsWorks para trabajar con pilas y aplicaciones
- Compatibilidad con otros servicios y configuraciones de AWS

Ejemplos de código con orientación para la AWS SDK for .NET

Las siguientes secciones contienen ejemplos de código y proporcionan orientación sobre esos ejemplos. Pueden ayudarlo a aprender a usarlo AWS SDK for .NET para trabajar con AWS los servicios.

Si eres nuevo en el tema AWS SDK for .NET, quizás quieras consultar primero el Recorrido rápido tema. que sirve de introducción a SDK.

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Temas

- Acceder AWS CloudFormation con el AWS SDK for .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 for .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 AWS CloudFormation con el AWS SDK for .NET

El AWS SDK for .NET soporte AWS CloudFormation, que crea y aprovisiona los despliegues de AWS infraestructura de forma predecible y repetida.

API

AWS SDK for .NET Proporciona API para AWS CloudFormation los clientes. Las API le permiten trabajar con AWS CloudFormation funciones como plantillas y pilas. Esta sección contiene un número reducido de ejemplos que muestran los patrones que se pueden seguir al usar estas API. Para ver el conjunto completo de API, consulta la Referencia de AWS SDK for .NET API (y desplázate hasta «Amazon». CloudFormation«).

Las AWS CloudFormation API las proporciona AWSSDK. CloudFormationpaquete.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Temas

Temas

- Enumerar AWS los recursos utilizando AWS CloudFormation

Enumerar AWS los recursos utilizando AWS CloudFormation

En este ejemplo, se muestra cómo utilizarlos AWS SDK for .NET para enumerar los recursos en AWS 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 I. CloudFormationPaginatorFactory 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($"\nIn Region:
{_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

Console.WriteLine("\n-----------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
```

```
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n---------------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                    " the AWSSDK.SSO and AWSSDK.SSOOIDC 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 for .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 for .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 autentiquen. 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 for .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 for .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 AmazonCognitoIdentityProviderClientcon 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 for .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.

En la siguiente información se presentan estos modelos y sus API, se proporcionan ejemplos acerca de cómo y cuándo usarlos y se facilitan enlaces a más recursos de programación de DynamoDB en AWS SDK for .NET.

Temas

Modelo de bajo nivel

El modelo de programación de bajo nivel incluye llamadas directas al servicio DynamoDB. Puede obtener acceso a este modelo a través del espacio de nombres Amazon.DynamoDBv2.

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.DynamoDBv2. DocumentModelespacio 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 método

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 for .NET.

TableBuilder

En la versión 3.7.203 del `TableBuilder`paquete AWSSDK NuGet .DynamoDBV2 se introdujo un mecanismo alternativo, la clase. 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 for .NET.

Inserción 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 Amazon.DynamoDBv2. DataModelespacio 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 for .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 NuGet paquete AWSSDK .DynamoDBv2, permite evitar determinadas condiciones que podrían provocar una latencia adicional o bloqueos debido a los comportamientos de arranque en frío y de grupo de subprocesos. Para obtener más información, consulte la entrada de blog Patrones de inicialización de DynamoDB mejorados para AWS SDK for .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 .DynamoDBv2. AWSSDK 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

Utilización de AWS SDK for .NET para programar información y ejemplos de DynamoDB**

- API de DynamoDB

- 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 for .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 for .NET
- Trabajar con elementos mediante la API de AWS SDK for .NET bajo nivel
- Consulta de tablas mediante la API de bajo nivel AWS SDK for .NET
- Escaneo de tablas mediante la API de bajo nivel AWS SDK for .NET
- Trabajar con índices secundarios locales mediante la API de bajo nivel AWS SDK for .NET
- Trabajar con índices secundarios globales mediante la API de bajo nivel AWS SDK for .NET

Información y ejemplos del modelo de documento

- Tipos de datos de DynamoDB
- DynamoDBEntry
- .NET: modelo de documento

Información y ejemplos del modelo de persistencia de objetos

- .NET: modelo de persistencia de objetos

Uso de expresiones con Amazon DynamoDB y AWS SDK for .NET

> **Note**
>
> La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK for .NET versión 3.3 y anteriores.

Los siguientes ejemplos de código muestran cómo utilizar el AWS SDK for .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

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 `BOOL`, `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": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      },
```

```
      {
        "M": {
          "RearView": {
            "S": "http://example/products/205_rear.jpg"
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/205_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "Excellent! Can't recommend it highly enough! Buy it!",
          "Do yourself a favor and buy this."
        ]
      },
      "OneStar": {
        "SS": [
          "Terrible product! Do not buy this."
        ]
      }
    }
  }
},
{
  "Id": {
    "N": "301"
  },
  "Title": {
    "S": "18-Bicycle 301"
  },
  "Description": {
    "S": "301 description"
  },
  "BicycleType": {
    "S": "Road"
```

```
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "185"
  },
  "Gender": {
    "S": "F"
  },
  "Color": {
    "SS": [
      "Blue",
      "Silver"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "3"
  },
  "RelatedItems": {
    "NS": [
      "801",
      "822",
      "979"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/301_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
```

```
            "S": "http://example/products/301_rear.jpg"
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/301_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "My daughter really enjoyed this bike!"
        ]
      },
      "ThreeStar": {
        "SS": [
          "This bike was okay, but I would have preferred it in my color.",
       "Fun to ride."
        ]
      }
    }
  }
}
```

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`,, y`SS`, 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 for .NET de bajo nivel
- Consulta de tablas mediante la API de bajo nivel AWS SDK for .NET
- Escaneo de tablas mediante la API de bajo nivel AWS SDK for .NET
- Trabajar con índices secundarios locales mediante la API de bajo nivel AWS SDK for .NET
- Trabajar con índices secundarios globales mediante la API de bajo nivel AWS SDK for .NET

Compatibilidad con JSON en Amazon DynamoDB

> **Note**
>
> La información de este tema es específica de los proyectos basados en .NET Framework y en la AWS SDK for .NET versión 3.3 y anteriores.

AWS SDK for .NET Admite datos de 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 for .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 for .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.

API

AWS SDK for .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 un número reducido de ejemplos que muestran los patrones que se pueden seguir al usar estas API. Para ver el conjunto de API completo, consulte la Referencia de API de AWS SDK for .NET (y desplácese a “Amazon.EC2”).

Las API de Amazon EC2 se proporcionan en el AWSSDK NuGet paquete.EC2.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de 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 los 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 is How to Prepare.

Para obtener información sobre las API y los 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 utilizarlos para AWS SDK for .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

  Clase AmazonEC2Client
- 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 DescribeSecurityGroupsRequestclase.

- 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 for .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-Classic si su AWS cuenta lo admite.

Si no proporciona un ID de VPC y su AWS cuenta no es compatible con EC2-Classic, el nuevo grupo de seguridad pertenecerá a la VPC predeterminada de su cuenta.

> **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 is how to prepare.

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

  Clase AmazonEC2Client
- 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($"\nGroupName: {group.GroupName}");
        Console.WriteLine($"GroupId: {group.GroupId}");
        Console.WriteLine($"Description: {group.Description}");
        Console.WriteLine($"VpcId (if any): {group.VpcId}");
      }
    }


    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already 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 for .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

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

  Clase AmazonEC2Client
- 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($"\nThe 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($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2AddRuleForRDP -g <group-id> -i <ip-address> [-p <port>]" +
        "\n  -g, --group-id: The ID of the security group to which you want to add the
 inbound rule." +
        "\n  -i, --ip-address: An IP address or address range in CIDR format." +
        "\n  -p, --port: The TCP port number. Defaults to 3389.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
 it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
```

```
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
```

```
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

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 IpPermissionclase.

- 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 Añadir 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 las API y los 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 for .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

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($"\nCreated new key pair: {response.KeyPair.KeyName}");

  // Save the private key in a PEM file
  using (var s = new FileStream(pemFileName, FileMode.Create))
  using (var writer = new StreamWriter(s))
  {
```

```
        writer.WriteLine(response.KeyPair.KeyMaterial);
      }
    }
```

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 (KeyPairInfo 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

  Clase AmazonEC2Client
- 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($"\nCreated new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
      writer.WriteLine(response.KeyPair.KeyMaterial);
    }
  }


  //
  // Method to show the key pairs that are available
  private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
  {
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyPairInfo 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 for .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 (KeyPairInfo 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

  Clase AmazonEC2Client
- 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($"\nKey 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 (KeyPairInfo 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 sobre las regiones y las zonas de disponibilidad, consulte Regiones y zonas en la Guía del usuario de Amazon EC2.

En este ejemplo, se muestra cómo utilizarlas AWS SDK for .NET para obtener detalles sobre las regiones y zonas de disponibilidad relacionadas con un cliente de 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

  Clase AmazonEC2Client
- 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 for .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 EC2, consulte las instancias de Amazon EC2 en la Guía del usuario de Amazon EC2.

Para obtener información sobre las API y los 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 for .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 is Retiring: Here is How to Prepare.

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

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 su 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 información sobre las AMI, consulte Amazon Machine Images (AMI) en la Guía del usuario de Amazon EC2. En concreto, consulte Buscar una AMI y AMI compartidas.

- 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 InstanceStateclase 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.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }
```

Código completo

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

  Clase AmazonEC2Client

  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 DescribeInstancesRequestobjeto. 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 RunInstancesRequestobjeto `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 quieres usar un tipo de instancia diferente al de este ejemplo, puedes elegir entre varios tipos de instancias. Para obtener más información, consulte los tipos de instancias de Amazon EC2 en la Guía del usuario de Amazon EC2. Consulte también los detalles del tipo de instancia y el explorador de tipos de instancia.

- También puede asociar un rol de IAM a una instancia al lanzarla. Para ello, cree un IamInstanceProfileSpecificationobjeto cuya `Name` propiedad tenga el nombre de un rol de IAM. A continuación, añada ese objeto a la `IamInstanceProfile` propiedad del RunInstancesRequestobjeto.

  > **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 Conceder un permiso de usuario para transferir una función 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 Conectarse a su 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 Conectarse a su 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

  Clase AmazonEC2Client
- 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($"\nPassword: {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($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"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 for .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

Clase AmazonEC2Client

- 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 for .NET para gestionar las instancias puntuales de Amazon EC2.

Información general

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, termínela, 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 puntuales, consulte las instancias puntuales en la Guía del usuario de Amazon EC2.

Acerca de este tutorial

Al seguir este tutorial, utilizará el AWS SDK for .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

Para obtener información sobre las API y los 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 entre los que elegir. Para obtener más información, consulte los tipos de instancias de Amazon EC2 en la Guía del usuario de Amazon EC2. Consulte también los detalles del tipo de instancia y el 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 información sobre las AMI, consulte Amazon Machine Images (AMI) en la Guía del usuario de Amazon EC2. En concreto, consulte Buscar una AMI y AMI compartidas.

- 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 puntuales 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 SpotInstanceRequestobjeto 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 puntuales, consulte el estado de las solicitudes puntuales 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

  Clase AmazonEC2Client

  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($"\nSpot 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 for .NET

The AWS SDK for .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.

API

AWS SDK for .NET Proporciona API para clientes de IAM. Estas API permiten trabajar con características de IAM, como usuarios, roles y claves de acceso.

Esta sección contiene un número reducido de ejemplos que muestran los patrones que se pueden seguir al usar estas API. Para ver el conjunto completo de API, consulta la Referencia de AWS SDK for .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.

Las API de IAM las AWSSDK proporciona. IdentityManagement NuGetpaquete.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de 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 for .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($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  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:

```
{
  "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 lisa 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 for .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 reponseVersions.Versions)
      {
        if(version.IsDefaultVersion)
        {
          defaultVersion = version.VersionId;
          break;
        }
      }
```

```
      return defaultVersion;
    }


    //
    // Method to retrieve and display the policy document of an IAM policy
    private static async Task ShowPolicyDocument(
      IAmazonIdentityManagementService iamClient, string policyArn, string
 defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
  }
}
```

Concesión de acceso mediante un rol de IAM

En este tutorial, se muestra cómo utilizarlos AWS SDK for .NET para habilitar las funciones de IAM en las instancias de Amazon EC2.

Información general

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 funciones de IAM, consulte Funciones 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 for .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 for .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 que utilice 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, consulteConfiguración de la autenticación de SDK con 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 lanzar una instancia mediante la consola EC2, consulte Lanzar una instancia mediante el asistente de lanzamiento de nuevas instancias en la Guía del usuario de Amazon EC2.

  Al revisar la página de lanzamiento, al menos debería ampliar el panel de detalles avanzados para poder especificar el rol de IAM que creó anteriormente en el perfil de la instancia de IAM.

- Usando el AWS SDK for .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 Conceder un permiso de usuario para transferir una función 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 información sobre cómo conectarse a una instancia, consulte Conectarse a su instancia de Linux o Conectarse a su 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 Conectarse a su instancia de Linux (consulte la subsección correspondiente) o Transferir 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 for .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.

API

AWS SDK for .NET Proporciona API para clientes de Amazon S3. Estas API permiten trabajar con recursos de Amazon S3, como buckets y elementos. Para ver el conjunto completo de API de Amazon S3, consulte lo siguiente:

- AWS SDK for .NET Referencia de API (y desplázate hasta «Amazon.S3").
- Documentación de Amazon.Extensions.S3.Encryption

Las API de Amazon S3 se proporcionan en los siguientes NuGet paquetes:

- AWSSDKS.3
- Amazon.Extensions.S3.Encryption

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Ejemplos en este documento

Los siguientes temas de este documento muestran cómo utilizarlos AWS SDK for .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 for .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 for .NET utiliza .NET Framework, también son válidos para versiones posteriores de la versión que AWS SDK for .NET utiliza .NET Core. A veces es necesario realizar pequeños ajustes en el código.

Ejemplos de programación de Amazon S3

- Administración de las ACL
- 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 for .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 for .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 de clientes de cifrado de S3.

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($"\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 for .NET es interoperable con la `AmazonS3EncryptionClientV2` clase de. AWS SDK for Java Esto significa que puede cifrar con el AWS SDK for Java y descifrar con el AWS SDK for .NET, y viceversa.

> **Note**
>
> La `AmazonS3EncryptionClientV2` clase de los AWS SDK for .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 for .NET es incompatible con la `AmazonS3EncryptionClientV2` clase de AWS SDK for 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 for .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 for .NET versión 3.3 y anteriores.

AWS SDK for .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.USWest2);

            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 for .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.

API

AWS SDK for .NET Proporciona API para los clientes de Amazon SQS. Estas API permiten trabajar con características de Amazon SQS como colas y mensajes. Esta sección contiene un número reducido de ejemplos que muestran los patrones que se pueden seguir al usar estas API. Para ver el conjunto de API completo, consulte la Referencia de API de AWS SDK for .NET (y desplácese a "Amazon.SQS").

Las API de Amazon SQS se proporcionan en el paquete AWSSDK NuGet .SQS.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de 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 for .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

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($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{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 AmazonSQSClient

  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($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
      "\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
      "\n  -q, --queue-name: The name of the queue you want to create." +
      "\n  -d, --dead-letter-queue: The URL of an existing queue to be used as the
 dead-letter queue."+
      "\n      If this argument isn't supplied, a new dead-letter queue will be
 created." +
      "\n  -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
 of the queue." +
      $"\n      Default is {MaxReceiveCount}." +
      "\n  -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
 for long polling." +
      $"\n      Default is {ReceiveMessageWaitTime}.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class that represents a command line on the console or terminal.
```

```
 // (This is the same for all examples. When you have seen it once, you can ignore
it.)
 static class CommandLine
 {
   //
   // Method to parse a command line of the form: "--key value" or "-k value".
   //
   // Parameters:
   // - args: The command-line arguments passed into the application by the system.
   //
   // Returns:
   // A Dictionary with string Keys and Values.
   //
   // If a key is found without a matching value, Dictionary.Value is set to the key
   //  (including the dashes).
   // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
   //  where "N" represents sequential numbers.
   public static Dictionary<string,string> Parse(string[] args)
   {
     var parsedArgs = new Dictionary<string,string>();
     int i = 0, n = 0;
     while(i < args.Length)
     {
       // If the first argument in this iteration starts with a dash it's an option.
       if(args[i].StartsWith("-"))
       {
         var key = args[i++];
         var value = key;

         // Check to see if there's a value that goes with this option?
         if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
         parsedArgs.Add(key, value);
       }

       // If the first argument in this iteration doesn't start with a dash, it's a
value
       else
       {
         parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
         n++;
       }
     }

     return parsedArgs;
```

```
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Consideraciones adicionales

- El nombre de la cola debe estar formado solo por caracteres alfanuméricos, guiones y guiones bajos.
- En los nombres y las URL de las colas se distingue entre mayúsculas y 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 CreateQueueRequestla AWS SDK for .NET API o la referencia de SetQueueAttributesla 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 se puede especificar un sondeo prolongado durante una llamada a los métodos `ReceiveMessageAsync` de la clase AmazonSQSClient. 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 for .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 AmazonSQSClient

  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 atributo `RedrivePolicy`, debe entrecomillar el valor completo y escapar las comillas en los pares clave/valor, según corresponda en 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 for .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

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 [AmazonSQSClient]
- 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 for .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($"\nSending a batch of messages to queue\n  {qUrl}");
      SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
      // Could test responseSendBatch.Failed here
      foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
    }
```

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($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"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 AmazonSQSClient
- 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 for .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

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 AmazonSQSClient
- 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, consulte los métodos `ChangeMessageVisibilityAsync` de la clase AmazonSQSClient.

- 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 de AWS Lambda como servicio de computación

AWS SDK for .NET admite AWS Lambda, que 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#.

API

AWS SDK for .NET proporciona API para AWS Lambda. Las API permiten trabajar con características de Lambda, como, por ejemplo, funciones, desencadenadores y eventos. Para ver el conjunto de API completo, consulte Lambda en Referencia de API de AWS SDK for .NET.

Las API de Lambda se proporcionan mediante paquetes NuGet.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Temas

Temas

- Uso de anotaciones para escribir funciones de AWS Lambda

Uso de anotaciones para escribir funciones de AWS Lambda

Al escribir funciones 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 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:

- README de GitHub, para obtener documentación sobre las API y los atributos de las Lambda Annotations.
- La entrada de blog de Lambda Annotations.
- El paquete NuGet `Amazon.Lambda.Annotations`.
- Proyecto PhotoAssetManagement en GitHub. En concreto, consulte la carpeta PamApiAnnotations y las referencias a Lambda Annotations del archivo README del proyecto.

Bibliotecas y marcos de alto nivel para AWS SDK for .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 no está familiarizado con AWS SDK for .NET, quizás le convenga echar un vistazo primero al tema Recorrido rápido, que sirve de introducción a SDK.

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Temas

- AWS Marco de procesamiento de mensajes para.NET

AWS Marco de procesamiento de mensajes para.NET

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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 comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de 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/awslabs/ 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

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de 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 EventBridgeguías para desarrolladores.
- 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, consulteInstalación y configuración de la cadena de herramientas.
- En este tutorial se supone 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 Configuración de la autenticación de SDK con 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 . AWS Tools for 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
```

AWS Tools for 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.SSOOIDC`.

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 de la . 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. Abre 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.SSOOIDC`.

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 de la . 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 AWS CLI o AWS Tools for PowerShell para eliminar la cola.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

AWS Tools for PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publica mensajes con el marco de procesamiento de AWS mensajes para.NET

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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 entre proyectos. Sin embargo, si los mensajes se redefinen en espacios de nombres diferentes o si intercambias 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 genéricoIMessagePublisher, 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 tiposISQSPublisher, y. ISNSPublisher 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 for .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

> Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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 que caduque, el marco ampliará el tiempo de espera de visibilidad (unos `VisibilityTimeout` segundos más). El valor predeterminado es 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Con qué frecuencia, en 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;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

Manejo de mensajes en 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

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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, consulteSeguridad del marco de procesamiento de AWS mensajes para.NET.

Personalice el marco de procesamiento de AWS mensajes para.NET

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

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 EC2; de lo contrario, será un valor alternativo de. `/aws/messaging`
  - Puede anular esto mediante `AddMessageSource` o en el. `AddMessageSourceSuffix` `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`son `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

Esta es una documentación preliminar para una característica en versión de vista previa. Está sujeta a cambios.

El marco de procesamiento de AWS mensajes de.NET se basa en el AWS SDK for .NET para comunicarse con AWS. Para obtener más información sobre la seguridad en AWS SDK for .NET, consulteSeguridad de 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.

Programación de AWS OpsWorks para trabajar con pilas y aplicaciones

Warning

AWS OpsWorks está llegando al final de su vida útil y no acepta nuevos clientes. Los clientes actuales no se verán afectados hasta marzo o mayo de 2024, según los servicios que utilicen, momento en el que el servicio dejará de estar disponible. A fin de tenerlo todo listo para esta transición, recomendamos que los clientes actuales migren a otras soluciones lo antes posible. Para obtener más información, consulte la página del producto de OpsWorks.

El AWS SDK for .NET es compatible con AWS OpsWorks, que ofrece una forma sencilla y flexible de crear y administrar pilas y aplicaciones. AWS OpsWorks permite aprovisionar recursos de AWS, administrar su configuración, implementar aplicaciones en dichos recursos y monitorizar su estado.

Para obtener más información, consulte la página del producto OpsWorks y la Guía del usuario de AWS OpsWorks.

API

AWS SDK for .NET proporciona API para AWS OpsWorks. Las API permiten trabajar con características de AWS OpsWorks como pilas con sus capas, instancias y aplicaciones. Para ver el conjunto de API completo, consulte la Referencia de API de AWS SDK for .NET (y desplácese a "Amazon.OpsWorks").

Las API de AWS OpsWorks se proporcionan mediante el paquete NuGet AWSSDK.OpsWorks.

Requisitos previos

Antes de comenzar, asegúrese de que ha configurado el entorno y el proyecto. Revise también la información en Características de SDK.

Compatibilidad con otros servicios y configuraciones de AWS

AWS SDK for .NET admite otros servicios de AWS aparte de los descritos en las secciones anteriores. Para obtener información sobre las API de todos los servicios compatibles, consulte la Referencia de API de AWS SDK for .NET.

Además de los espacios de nombres de servicios de AWS individuales, AWS SDK for .NET también proporciona las siguientes API:

Área	Descripción	Recursos
Compatibilidad con AWS	Acceso mediante programación a casos de AWS Support y características de Trusted Advisor.	Consulte Amazon.AWSSupport y Amazon.AWSSupport.Model.
General	Clases auxiliares y enumeraciones.	Consulte Amazon y Amazon.Util.

AWS SDK for .NET ejemplos de código

Los ejemplos de código de este tema muestran cómo usar el AWS SDK for .NET with 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 funciones de servicio individuales, 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.

Servicios

- ACMejemplos que utilizan AWS SDK for .NET
- APIEjemplos de puertas de enlace que utilizan AWS SDK for .NET
- Ejemplos de Aurora usando AWS SDK for .NET
- Ejemplos de Auto Scaling usando AWS SDK for .NET
- Ejemplos de Amazon Bedrock que utilizan AWS SDK for .NET
- Ejemplos de Amazon Bedrock Runtime que utilizan AWS SDK for .NET
- AWS CloudFormation ejemplos que utilizan AWS SDK for .NET
- CloudWatch ejemplos que utilizan AWS SDK for .NET
- CloudWatch Registra ejemplos usando AWS SDK for .NET
- Ejemplos de Amazon Cognito Identity Provider que utilizan AWS SDK for .NET
- Amazon Comprehend ejemplos utilizando AWS SDK for .NET
- Ejemplos de DynamoDB que utilizan AWS SDK for .NET
- EC2Ejemplos de Amazon que utilizan AWS SDK for .NET
- ECSEjemplos de Amazon que utilizan AWS SDK for .NET
- Ejemplos de Elastic Load Balancing: versión 2 utilizando AWS SDK for .NET
- EventBridge ejemplos que utilizan AWS SDK for .NET
- EventBridge Ejemplos de planificadores que utilizan AWS SDK for .NET

- AWS Glue ejemplos que utilizan AWS SDK for .NET
- IAMejemplos que utilizan AWS SDK for .NET
- Ejemplos de Amazon Keyspaces que utilizan AWS SDK for .NET
- Ejemplos de Kinesis que utilizan AWS SDK for .NET
- AWS KMS ejemplos que utilizan AWS SDK for .NET
- Ejemplos de Lambda que utilizan AWS SDK for .NET
- MediaConvert ejemplos que utilizan AWS SDK for .NET
- MSKEjemplos de Amazon que utilizan AWS SDK for .NET
- Ejemplos de organizaciones que utilizan AWS SDK for .NET
- Ejemplos de Amazon Pinpoint con AWS SDK for .NET
- Ejemplos de Amazon Polly que utilizan AWS SDK for .NET
- RDSEjemplos de Amazon que utilizan AWS SDK for .NET
- Ejemplos RDS de Amazon Data Service que utilizan AWS SDK for .NET
- Ejemplos de Amazon Rekognition que utilizan AWS SDK for .NET
- Ejemplos de registro de dominios de Route 53 utilizando AWS SDK for .NET
- Ejemplos de Amazon S3 que utilizan AWS SDK for .NET
- Ejemplos de S3 Glacier utilizando AWS SDK for .NET
- SageMaker ejemplos que utilizan AWS SDK for .NET
- Ejemplos de Secrets Manager usando AWS SDK for .NET
- SESEjemplos de Amazon que utilizan AWS SDK for .NET
- Ejemplos de Amazon SES API v2 que utilizan AWS SDK for .NET
- SNSEjemplos de Amazon que utilizan AWS SDK for .NET
- SQSEjemplos de Amazon que utilizan AWS SDK for .NET
- Ejemplos de Step Functions usando AWS SDK for .NET
- AWS STS ejemplos que utilizan AWS SDK for .NET
- AWS Support ejemplos que utilizan AWS SDK for .NET
- Ejemplos de Amazon Textract usando AWS SDK for .NET
- Ejemplos de Amazon Transcribe utilizando AWS SDK for .NET
- Ejemplos de Amazon Translate utilizando AWS SDK for .NET

ACMejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET withACM.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones

Acciones

`DescribeCertificate`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeCertificate`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

namespace DescribeCertificate
{
    class DescribeCertificate
    {
        // The following example retrieves and displays the metadata for a
```

```
        // certificate using the AWS Certificate Manager (ACM) service.

        // Specify your AWS Region (an example Region is shown).
        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new
 Amazon.CertificateManager.AmazonCertificateManagerClient(ACMRegion);

            var describeCertificateReq = new DescribeCertificateRequest();
            // The ARN used here is just an example. Replace it with the ARN of
            // a certificate that exists on your account.
            describeCertificateReq.CertificateArn =
                "arn:aws:acm:us-
east-1:123456789012:certificate/8cfd7dae-9b6a-2d07-92bc-1c309EXAMPLE";

            var certificateDetailResp =
                DescribeCertificateResponseAsync(client: _client, request:
 describeCertificateReq);
            var certificateDetail = certificateDetailResp.Result.Certificate;

            if (certificateDetail is not null)
            {
                DisplayCertificateDetails(certificateDetail);
            }
        }

        /// <summary>
        /// Displays detailed metadata about a certificate retrieved
        /// using the ACM service.
        /// </summary>
        /// <param name="certificateDetail">The object that contains details
        /// returned from the call to DescribeCertificateAsync.</param>
        static void DisplayCertificateDetails(CertificateDetail certificateDetail)
        {
            Console.WriteLine("\nCertificate Details: ");
            Console.WriteLine($"Certificate Domain:
 {certificateDetail.DomainName}");
            Console.WriteLine($"Certificate Arn:
 {certificateDetail.CertificateArn}");
            Console.WriteLine($"Certificate Subject: {certificateDetail.Subject}");
            Console.WriteLine($"Certificate Status: {certificateDetail.Status}");
```

```
            foreach (var san in certificateDetail.SubjectAlternativeNames)
            {
                Console.WriteLine($"Certificate SubjectAlternativeName: {san}");
            }
        }

        /// <summary>
        /// Retrieves the metadata associated with the ACM service certificate.
        /// </summary>
        /// <param name="client">An AmazonCertificateManagerClient object
        /// used to call DescribeCertificateResponse.</param>
        /// <param name="request">The DescribeCertificateRequest object that
        /// will be passed to the method call.</param>
        /// <returns></returns>
        static async Task<DescribeCertificateResponse>
 DescribeCertificateResponseAsync(
            AmazonCertificateManagerClient client, DescribeCertificateRequest
 request)
        {
            var response = new DescribeCertificateResponse();

            try
            {
                response = await client.DescribeCertificateAsync(request);
            }
            catch (InvalidArnException)
            {
                Console.WriteLine($"Error: The ARN specified is invalid.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Error: The specified certificate could not be
 found.");
            }

            return response;
        }
    }

}
```

- Para API obtener más información, consulte DescribeCertificatela AWS SDK for .NET APIReferencia.

`ListCertificates`

En el siguiente ejemplo de código, se muestra cómo usar `ListCertificates`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

namespace ListCertificates
{
    // The following example retrieves and displays a list of the
    // certificates defined for the default account using the AWS
    // Certificate Manager (ACM) service.
    class ListCertificates
    {
        // Specify your AWS Region (an example Region is shown).

        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new AmazonCertificateManagerClient(ACMRegion);
            var certificateList = ListCertificatesResponseAsync(client: _client);

            Console.WriteLine("Certificate Summary List\n");
```

```
            foreach (var certificate in
certificateList.Result.CertificateSummaryList)
            {
                Console.WriteLine($"Certificate Domain: {certificate.DomainName}");
                Console.WriteLine($"Certificate ARN:
{certificate.CertificateArn}\n");
            }
        }

        /// <summary>
        /// Retrieves a list of the certificates defined in this Region.
        /// </summary>
        /// <param name="client">The ACM client object passed to the
        /// ListCertificateResAsync method call.</param>
        /// <param name="request"></param>
        /// <returns>The ListCertificatesResponse.</returns>
        static async Task<ListCertificatesResponse> ListCertificatesResponseAsync(
            AmazonCertificateManagerClient client)
        {
            var request = new ListCertificatesRequest();

            var response = await client.ListCertificatesAsync(request);
            return response;
        }
    }
}
```

- Para API obtener más información, consulte ListCertificatesla AWS SDK for .NET APIReferencia.

APIEjemplos de puertas de enlace que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for .NET API Gateway.

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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Escenarios

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

> Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.
>
> Para obtener el código fuente completo e instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en GitHub.
>
> Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.
>
> Servicios utilizados en este ejemplo
>
> - APIPuerta de enlace
> - DynamoDB
> - Lambda
> - Amazon Rekognition
> - Amazon S3
> - Amazon SNS

Ejemplos de Aurora usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso AWS SDK for .NET de 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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Aurora

En el siguiente ejemplo de código se muestra cómo empezar a utilizar Aurora.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\tCluster: database: {cluster.DatabaseName}
 identifier: {cluster.DBClusterIdentifier}.");
        }
    }
}
```

- Para API obtener más información, consulte D escribeDBClusters en la AWS SDK for .NET APIreferencia.

Temas

- Conceptos básicos
- Acciones
- Escenarios

Conceptos básicos

Aprenda los 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.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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(
                $"\t{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(
                $"\n\tParameter: {p.ParameterName}." +
```

```
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> 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(
                $"\t{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 && 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.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
```

```
    /// 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;
    }

    /// <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);
            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="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 API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - C reateDBCluster
  - C reateDBCluster ParameterGroup
  - reateDBClusterInstantánea C
  - C reateDBInstance
  - D eleteDBCluster
  - D eleteDBCluster ParameterGroup
  - D eleteDBInstance
  - D escribeDBCluster ParameterGroups
  - escribeDBClusterParámetros D
  - escribeDBClusterInstantáneas en 3D
  - D escribeDBClusters
  - escribeDBEngineVersiones D
  - D escribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - M odifyDBCluster ParameterGroup

Acciones

`CreateDBCluster`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBCluster`.

AWS SDK for .NET

> **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 API obtener más información, consulte C reateDBCluster en la AWS SDK for .NET APIreferencia.

`CreateDBClusterParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBClusterParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte C reateDBCluster ParameterGroup en la AWS SDK for .NET APIreferencia.

`CreateDBClusterSnapshot`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBClusterSnapshot`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte C reateDBCluster Snapshot in AWS SDK for .NET APIReference.

`CreateDBInstance`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBInstance`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="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;
    }
```

- Para API obtener más información, consulte C reateDBInstance en la AWS SDK for .NET APIreferencia.

`DeleteDBCluster`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDBCluster`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte D eleteDBCluster en la AWS SDK for .NET APIreferencia.

`DeleteDBClusterParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDBClusterParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte D eleteDBCluster ParameterGroup en la AWS SDK for .NET APIreferencia.

`DeleteDBInstance`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDBInstance`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte D eleteDBInstance en la AWS SDK for .NET APIreferencia.

`DescribeDBClusterParameterGroups`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBClusterParameterGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte D escribeDBCluster ParameterGroups en la AWS SDK for .NET APIreferencia.

`DescribeDBClusterParameters`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBClusterParameters`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte escribeDBClusterlos parámetros D en AWS SDK for .NET APIla referencia.

`DescribeDBClusterSnapshots`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBClusterSnapshots`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte escribeDBClusterInstantáneas en D en AWS SDK for .NET APIreferencia.

`DescribeDBClusters`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBClusters`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre. 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);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Para API obtener más información, consulte D escribeDBClusters en la AWS SDK for .NET APIreferencia.

`DescribeDBEngineVersions`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBEngineVersions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte escribeDBEnginelas versiones D en AWS SDK for .NET APIla referencia.

`DescribeDBInstances`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte D escribeDBInstances en la AWS SDK for .NET APIreferencia.

`DescribeOrderableDBInstanceOptions`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeOrderableDBInstanceOptions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DescribeOrderableDBInstanceOptionsla AWS SDK for .NET APIReferencia.

`ModifyDBClusterParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `ModifyDBClusterParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 && 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 API obtener más información, consulte M odifyDBCluster ParameterGroup en la AWS SDK for .NET APIreferencia.

Escenarios

Crear un rastreador de elementos de trabajo de Aurora Serverless

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos Amazon Aurora Serverless y utilice Amazon Simple Email Service (AmazonSES) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizarla AWS SDK for .NET para crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos de Amazon Aurora y envíe informes por correo electrónico mediante Amazon Simple Email Service (AmazonSES). En este ejemplo, se utiliza una interfaz creada con React.js para interactuar con unRESTful. NETbackend.

- Integre una aplicación web de React con AWS los servicios.
- Muestre, agregue, actualice y elimine elementos en una tabla de Aurora.
- Envía un informe por correo electrónico de los artículos de trabajo filtrados a través de AmazonSES.
- Implemente y gestione recursos de ejemplo con el AWS CloudFormation script incluido.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Ejemplos de Auto Scaling usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el AWS SDK for .NET uso de 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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción al escalado automático

En los siguientes ejemplos de código se muestra cómo empezar a utilizar el escalado automático.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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();

        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availability
```

```
        });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
        }
    }
}
```

- Para API obtener más información, consulte DescribeAutoScalingGroupsla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones
- Escenarios

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Cree un grupo de Amazon EC2 Auto Scaling con una plantilla de lanzamiento y zonas de disponibilidad, y obtenga 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.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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.First().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 is not null)
        {
            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 is not null)
        {
            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!);
        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!);
        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 ScalingEC2, Amazon 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
        };
```

```
        var response = await
 _amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <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<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };
```

```
        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var 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 groupList = new List<string>
            {
                groupName,
            };

        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        var 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<Amazon.CloudWatch.Model.Metric>>
 GetCloudWatchMetricsAsync(string groupName)
    {
        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);

        return response.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 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);

        return response.Datapoints;
    }

}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Acciones

`AttachLoadBalancerTargetGroups`

En el siguiente ejemplo de código, se muestra cómo usar `AttachLoadBalancerTargetGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
```

- Para API obtener más información, consulte AttachLoadBalancerTargetGroupsla AWS SDK for .NET APIReferencia.

`CreateAutoScalingGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateAutoScalingGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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
        };

        var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte CreateAutoScalingGroupla AWS SDK for .NET APIReferencia.

`DeleteAutoScalingGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteAutoScalingGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Actualice el tamaño mínimo de un grupo de escalado automático a cero, finalice todas las instancias del grupo y elimine el grupo.

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
```

```
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
```

```
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }
```

```
    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }
```

- Para API obtener más información, consulte DeleteAutoScalingGroupla AWS SDK for .NET APIReferencia.

`DescribeAutoScalingGroups`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAutoScalingGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
```

```
            InstanceIds = instanceIds,
        };

        var response = await
_amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- Para API obtener más información, consulte DescribeAutoScalingGroupsla AWS SDK for .NET APIReferencia.

`DescribeAutoScalingInstances`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAutoScalingInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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>();
        groups!.ForEach(group =>
```

```
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
_amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- Para API obtener más información, consulte DescribeAutoScalingInstancesla AWS SDK for .NET APIReferencia.

`DescribeScalingActivities`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeScalingActivities`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.Activities;
    }
```

- Para API obtener más información, consulte DescribeScalingActivitiesla AWS SDK for .NET APIReferencia.

`DisableMetricsCollection`

En el siguiente ejemplo de código, se muestra cómo usar `DisableMetricsCollection`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DisableMetricsCollectionla AWS SDK for .NET APIReferencia.

`EnableMetricsCollection`

En el siguiente ejemplo de código, se muestra cómo usar `EnableMetricsCollection`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte EnableMetricsCollectionla AWS SDK for .NET APIReferencia.

`SetDesiredCapacity`

En el siguiente ejemplo de código, se muestra cómo usar `SetDesiredCapacity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte SetDesiredCapacityla AWS SDK for .NET APIReferencia.

`TerminateInstanceInAutoScalingGroup`

En el siguiente ejemplo de código, se muestra cómo usar `TerminateInstanceInAutoScalingGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte TerminateInstanceInAutoScalingGroupla AWS SDK for .NET APIReferencia.

`UpdateAutoScalingGroup`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateAutoScalingGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte UpdateAutoScalingGroupla AWS SDK for .NET APIReferencia.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (AmazonEC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias en un rango específico.
- Gestione y distribuya HTTP las solicitudes con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecuta un servidor web Python en cada EC2 instancia para gestionar HTTP las solicitudes. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

AWS SDK for .NET

> **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 el escenario interactivo en un símbolo del sistema.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
```

```
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }
```

```
   /// <summary>
   /// Deploy necessary resources for the scenario.
   /// </summary>
   /// <param name="interactive">True to run as interactive.</param>
   /// <returns>True if successful.</returns>
   public static async Task<bool> Deploy(bool interactive)
   {
       var protocol = "HTTP";
       var port = 80;
       var sshPort = 22;

       Console.WriteLine(
           "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
           "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
           "against various kinds of failures.\n\n" +
           "Some of the resources create by this demo are:\n");

       Console.WriteLine(
           "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
       Console.WriteLine(
           "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
       Console.WriteLine(
           "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
       Console.WriteLine(
           "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
       Console.WriteLine(new string('-', 80));
       Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
       if (interactive)
           Console.ReadLine();

       // Create and populate the DynamoDB table.
       var databaseTableName = _configuration["databaseName"];
       var recommendationsPath = Path.Join(_configuration["resourcePath"],
           "recommendations_objects.json");
       Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
```

```
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
```

```
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);

        await
_elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
subnetIds, targetGroup);
        await
_autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
_elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
```

```
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
 ipString);
                }
```

```
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();
```

```
        Console.WriteLine("\nThis part of the demonstration shows how to toggle
 different parts of the system\n" +
                          "to create situations where the web service fails, and
 shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
 these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
 recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
 recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
 the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
 customer with a somewhat better experience than failure.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
 "static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
 checks are still shallow.");
        if (interactive)
            await DemoActionChoices();
```

```
        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
```

```
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");
```

```
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
 can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
 resources? (y/n) "))
        {
            await
 _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
```

```
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Crea una clase que agrupe las EC2 acciones de Auto Scaling y Amazon.

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
```

```
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
```

```
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";
```

```
        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
```

```
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
```

```
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
```

```
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
```

```
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
```

```
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
```

```
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
```

```
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }
```

```
   /// <summary>
   /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
   /// </summary>
   /// <param name="group">The name of the auto scaling group.</param>
   /// <returns>A collection of instance Ids.</returns>
   public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
   {
       var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
           new DescribeAutoScalingGroupsRequest()
           {
               AutoScalingGroupNames = new List<string>() { group }
           });
       var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
           g => g.Instances.Select(i => i.InstanceId));
       return instanceIds;
   }

   /// <summary>
   /// Get the instance profile association data for an instance.
   /// </summary>
   /// <param name="instanceId">The Id of the instance.</param>
   /// <returns>Instance profile associations data.</returns>
   public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
   {
       var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
           new DescribeIamInstanceProfileAssociationsRequest()
           {
               Filters = new List<Amazon.EC2.Model.Filter>()
               {
                   new ("instance-id", new List<string>() { instanceId })
               },
           });
       return response.IamInstanceProfileAssociations[0];
   }

   /// <summary>
   /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
   /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
```

```
   /// used to restart the Python web server.
   /// </summary>
   /// <param name="instanceId">The Id of the instance to update.</param>
   /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
   /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
                   AssociationId = associationId,
                   IamInstanceProfile = new IamInstanceProfileSpecification()
                   {
                       Name = credsProfileName
                   }
               });
       // Allow time before resetting.
       Thread.Sleep(25000);
       var instanceReady = false;
       var retries = 5;
       while (retries-- > 0 && !instanceReady)
       {
           await _amazonEc2.RebootInstancesAsync(
               new RebootInstancesRequest(new List<string>() { instanceId }));
           Thread.Sleep(10000);

           var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
               new DescribeInstanceInformationRequest());
           // Get the entire list using the paginator.
           await foreach (var instance in
instancesPaginator.InstanceInformationList)
           {
               instanceReady = instance.InstanceId == instanceId;
               if (instanceReady)
               {
                   break;
               }
           }
       }
```

```
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

```
    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
```

```
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }
```

```
   /// <summary>
   /// Verify the default security group of a Vpc allows ingress from the calling
computer.
   /// This can be done by allowing ingress from this computer's IP address.
   /// In some situations, such as connecting from a corporate network, you must
instead specify
   /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
   /// If you do, be sure to remove public access when you're done.
   /// </summary>
   /// <param name="vpc">The group to check.</param>
   /// <param name="port">The port to verify.</param>
   /// <param name="ipAddress">This computer's IP address.</param>
   /// <returns>True if the ip address is allowed on the group.</returns>
   public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
   {
       var portIsOpen = false;
       foreach (var ipPermission in group.IpPermissions)
       {
           if (ipPermission.FromPort == port)
           {
               foreach (var ipRange in ipPermission.Ipv4Ranges)
               {
                   var cidr = ipRange.CidrIp;
                   if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                   {
                       portIsOpen = true;
                   }
               }

               if (ipPermission.PrefixListIds.Any())
               {
                   portIsOpen = true;
               }

               if (!portIsOpen)
               {
                   Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
               }
               else
```

```
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
Scaling group.
    /// The
```

```
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Cree una clase que resuma las acciones de Elastic Load Balancing.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
```

```
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
```

```
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
```

```
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
 subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
 List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
 _amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
```

```
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
```

```
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });
```

```
                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
 {groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

Cree una clase que utilice DynamoDB para simular un servicio de recomendaciones.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
```

```
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
```

```
                            KeyType = KeyType.RANGE
                        }
                    },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
_amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
```

```
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Cree una clase que agrupe las acciones de Systems Manager.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
```

```
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations

- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Ejemplos de Amazon Bedrock que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante 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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon Bedrock

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon Bedrock.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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 ListFoundationModelsExample
{
    /// <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
            {
                ListFoundationModelsResponse 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 API obtener más información, consulte ListFoundationModelsla AWS SDK for .NET APIReferencia.

Temas

- Acciones

Acciones

`ListFoundationModels`

En el siguiente ejemplo de código, se muestra cómo usar `ListFoundationModels`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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
            {
                ListFoundationModelsResponse 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 API obtener más información, consulte ListFoundationModelsla AWS SDK for .NET APIReferencia.

Ejemplos de Amazon Bedrock Runtime que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante Amazon Bedrock Runtime. AWS SDK for .NET

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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Escenarios
- AI21Laboratorios: Jurassic-2
- Texto de Amazon Titan
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Escenarios

Crear una aplicación de sitio de pruebas que interactúe con modelos fundacionales de Amazon Bedrock

En el siguiente ejemplo de código se muestra cómo crear sitios de pruebas que interactúan con modelos fundacionales de Amazon Bedrock a través de diferentes modalidades.

AWS SDK for .NET

. NETFoundation Model (FM) Playground es un. NETMAUIAplicación de ejemplo de Blazor que muestra cómo usar Amazon Bedrock a partir del código C#. En este ejemplo se muestra cómo hacerlo. NETy los desarrolladores de C# pueden usar Amazon Bedrock para crear aplicaciones generativas habilitadas para la IA. Puede probar los modelos fundacionales de Amazon Bedrock e interactuar con ellos mediante los cuatro sitios de pruebas siguientes:

- Un sitio de pruebas de texto.
- Un sitio de pruebas de chat.
- Un sitio de pruebas de voz.
- Un sitio de pruebas de imágenes.

En el ejemplo también se enumeran y muestran los modelos fundacionales a los que tiene acceso y sus características. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en. GitHub

Servicios utilizados en este ejemplo

- Amazon Bedrock Runtime

AI21Laboratorios: Jurassic-2

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a AI21 Labs Jurassic-2 mediante Converse de Bedrock. API

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a AI21 Labs Jurassic-2, usando Converse de Bedrock. API

```
// Use the Converse API to send a text message to AI21 Labs Jurassic-2.

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., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// 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 API más información, consulte Converse in Reference.AWS SDK for .NET API

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a AI21 Labs Jurassic-2 mediante el modelo Invoke. API

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to AI21 Labs Jurassic-2.
```

```
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., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// 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
{
    prompt = userMessage,
    maxTokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["completions"]?[0]?["data"]?["text"] ?? "";
    Console.WriteLine(responseText);
```

```
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

Texto de Amazon Titan

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text con Converse API de Bedrock.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Amazon Titan Text con Converse API de Bedrock.

```
// Use the Converse API to send a text message to Amazon Titan Text.

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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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 API obtener más información, consulte Converse in Reference.AWS SDK for .NET API

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text mediante Converse de Bedrock API y cómo procesar el flujo de respuestas en tiempo real.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Amazon Titan Text con Converse de Bedrock API y procesa el flujo de respuestas en tiempo real.

```
// Use the Converse API to send a text message to Amazon Titan Text
// 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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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 API obtener más información, consulte ConverseStreamla referencia.AWS SDK for .NET API

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Amazon Titan Text mediante el modelo API Invoke.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Amazon Titan Text.

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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
```

```
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["results"]?[0]?["outputText"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos de Amazon Titan Text mediante el modelo API Invoke e imprimir el flujo de respuestas.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Amazon Titan Text
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
    }
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["outputText"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModelWithResponseStreamla AWS SDK for .NET APIReferencia.

Anthropic Claude

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude con Converse de Bedrock. API

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Anthropic Claude, usando Converse de Bedrock. API

```
// 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 API obtener más información, consulte Converse in Reference.AWS SDK for .NET API

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude mediante Converse de Bedrock API y procesar el flujo de respuestas en tiempo real.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Anthropic Claude con Converse de Bedrock API y procesa el flujo de respuestas 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 API más información, consulte la Referencia ConverseStream.AWS SDK for .NET API

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Anthropic Claude mediante el modelo Invoke. API

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API 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 API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos Anthropic Claude, utilizando el modelo InvokeAPI, e imprimir el flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Anthropic Claude
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
```

```
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["delta"]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
```

```
    throw;
}
```

- Para API obtener más información, consulte InvokeModelWithResponseStreamla AWS SDK for .NET APIReferencia.

Cohere Command

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante Converse de Bedrock. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Cohere Command, usando Converse de Bedrock. API

```
// 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 API obtener más información, consulte Converse in Reference.AWS SDK for .NET API

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante Converse de Bedrock API y procesar el flujo de respuestas en tiempo real.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Cohere Command con Converse de Bedrock API y procesa el flujo de respuestas 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 API más información, consulte la Referencia ConverseStream.AWS SDK for .NET API

InvokeModel: Comando R y R+

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command R y R + mediante el modelo Invoke. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Cohere Command R.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModel: Luz de mando y mando

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante el modelo Invoke. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Cohere Command.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command Light.
var modelId = "cohere.command-light-text-v14";

// 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
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);
```

```
    // Extract and print the response text.
    var responseText = modelResponse["generations"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream: Comando R y R+

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto al comando Cohere mediante el modelo Invoke API con un flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Cohere Command R
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
```

```
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream: Luz de mando y mando

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Cohere Command mediante el modelo Invoke API con un flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Cohere Command
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command Light.
var modelId = "cohere.command-light-text-v14";

// 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
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generations"]?[0]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

Meta Llama

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama con Converse de Bedrock. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Meta Llama, usando Converse API 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 API obtener más información, consulte Converse in Reference.AWS SDK for .NET API

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama utilizando Converse de Bedrock API y procesar el flujo de respuestas en tiempo real.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Meta Llama con Converse de Bedrock API y procesa el flujo de respuestas 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 API obtener más información, consulte ConverseStreamla Referencia.AWS SDK for .NET API

InvokeModel: Llama 2

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 2, utilizando el modelo API Invoke.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Meta Llama 2.

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., Llama 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
```

```
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["generation"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModel: Llama 3

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 3, utilizando el modelo API Invoke.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Meta Llama 3.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $@"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};
```

```
try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["generation"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream: Llama 2

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 2, utilizando el modelo InvokeAPI, e imprimir el flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Meta Llama 2
// and print the response stream.
```

```
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
```

```
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModelWithResponseStreamla AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream: Llama 3

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Meta Llama 3, utilizando el modelo InvokeAPI, e imprimir el flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Meta Llama 3
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
```

```
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $@"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);
```

```
    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModelWithResponseStreamla AWS SDK for .NET APIReferencia.

Mistral AI

Conversar

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral con Converse de Bedrock. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Mistral, usando Converse de Bedrock. API

```
// 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 API más información, consulte Converse in Reference.AWS SDK for .NET API

ConverseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a Mistral mediante Converse de Bedrock API y procesar el flujo de respuestas en tiempo real.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Envía un mensaje de texto a Mistral con Converse de Bedrock API y procesa el flujo de respuestas 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 API más información, consulte la Referencia. ConverseStreamAWS SDK for .NET API

InvokeModel

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos Mistral mediante el modelo Invoke. API

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa el modelo Invoke API para enviar un mensaje de texto.

```
// Use the native inference API to send a text message to Mistral.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";
```

```
//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["outputs"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModella AWS SDK for .NET APIReferencia.

InvokeModelWithResponseStream

El siguiente ejemplo de código muestra cómo enviar un mensaje de texto a los modelos Mistral AI, utilizando el modelo InvokeAPI, e imprimir el flujo de respuesta.

AWS SDK for .NET

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilice el modelo Invoke API para enviar un mensaje de texto y procesar el flujo de respuestas en tiempo real.

```
// Use the native inference API to send a text message to Mistral
// and print the response stream.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["outputs"]?[0]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para API obtener más información, consulte InvokeModelWithResponseStreamla AWS SDK for .NET APIReferencia.

AWS CloudFormation ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with AWS CloudFormation.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola AWS CloudFormation

En el siguiente ejemplo de código se muestra cómo empezar a utilizar AWS CloudFormation.

AWS SDK for .NET

> **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;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
```

```
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

Console.WriteLine("\n-----------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                 .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n-----------------------------------------------");
            return true;
        }
```

```
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                    " the AWSSDK.SSO and AWSSDK.SSOOIDC 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;
        }
    }
}
```

- Para API obtener más información, consulte DescribeStackResourcesla AWS SDK for .NET APIReferencia.

CloudWatch ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with 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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola CloudWatch

Los siguientes ejemplos de código muestran cómo empezar a usarlo CloudWatch.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListMetricsla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los 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.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha. 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($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{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($"\t{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($"\t{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($"\t{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() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {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++)
        {
            for (int j = 0; j < metricData[i].Values.Count; j++)
            {
                Console.WriteLine(
                    $"\tTimestamp {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
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo 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($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\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($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning 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;
    }

    /// <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;
    }


    /// <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}";
        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,
            });

        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;
    }

    /// <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;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - 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 usar `DeleteAlarms`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DeleteAlarmsla AWS SDK for .NET APIReferencia.

`DeleteAnomalyDetector`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteAnomalyDetector`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DeleteAnomalyDetectorla AWS SDK for .NET APIReferencia.

`DeleteDashboards`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDashboards`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DeleteDashboardsla AWS SDK for .NET APIReferencia.

`DescribeAlarmHistory`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAlarmHistory`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DescribeAlarmHistorylа AWS SDK for .NET APIReferencia.

`DescribeAlarms`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAlarms`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DescribeAlarmsla AWS SDK for .NET APIReferencia.

`DescribeAlarmsForMetric`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAlarmsForMetric`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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;
    }
```

- Para API obtener más información, consulte DescribeAlarmsForMetricla AWS SDK for .NET APIReferencia.

`DescribeAnomalyDetectors`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAnomalyDetectors`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DescribeAnomalyDetectorsla AWS SDK for .NET APIReferencia.

`DisableAlarmActions`

En el siguiente ejemplo de código, se muestra cómo usar `DisableAlarmActions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte DisableAlarmActionsla AWS SDK for .NET APIReferencia.

`EnableAlarmActions`

En el siguiente ejemplo de código, se muestra cómo usar `EnableAlarmActions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte EnableAlarmActionsla AWS SDK for .NET APIReferencia.

`GetDashboard`

En el siguiente ejemplo de código, se muestra cómo usar `GetDashboard`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte GetDashboardla AWS SDK for .NET APIReferencia.

`GetMetricData`

En el siguiente ejemplo de código, se muestra cómo usar `GetMetricData`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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}";
        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,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

- Para API obtener más información, consulte GetMetricDatala AWS SDK for .NET APIReferencia.

`GetMetricStatistics`

En el siguiente ejemplo de código, se muestra cómo usar `GetMetricStatistics`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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;
    }
```

- Para API obtener más información, consulte GetMetricStatisticsla AWS SDK for .NET APIReferencia.

`GetMetricWidgetImage`

En el siguiente ejemplo de código, se muestra cómo usar `GetMetricWidgetImage`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte GetMetricWidgetImagela AWS SDK for .NET APIReferencia.

`ListDashboards`

En el siguiente ejemplo de código, se muestra cómo usar `ListDashboards`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte ListDashboardsla AWS SDK for .NET APIReferencia.

`ListMetrics`

En el siguiente ejemplo de código, se muestra cómo usar `ListMetrics`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte ListMetricsla AWS SDK for .NET APIReferencia.

`PutAnomalyDetector`

En el siguiente ejemplo de código, se muestra cómo usar `PutAnomalyDetector`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte PutAnomalyDetectorla AWS SDK for .NET APIReferencia.

`PutDashboard`

En el siguiente ejemplo de código, se muestra cómo usar `PutDashboard`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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;
    }
```

- Para API obtener más información, consulte PutDashboardla AWS SDK for .NET APIReferencia.

`PutMetricAlarm`

En el siguiente ejemplo de código, se muestra cómo usar `PutMetricAlarm`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte PutMetricAlarmla AWS SDK for .NET APIReferencia.

`PutMetricData`

En el siguiente ejemplo de código, se muestra cómo usar `PutMetricData`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte PutMetricDatala AWS SDK for .NET APIReferencia.

CloudWatch Registra ejemplos usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso AWS SDK for .NET de 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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones

Acciones

`AssociateKmsKey`

En el siguiente ejemplo de código, se muestra cómo usar `AssociateKmsKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to associate an AWS Key Management Service (AWS KMS) key with
    /// an Amazon CloudWatch Logs log group.
    /// </summary>
    public class AssociateKmsKey
    {
        public static async Task Main()
        {
            // This client object will be associated with the same AWS Region
            // as the default user on this system. If you need to use a
            // different AWS Region, pass it as a parameter to the client
            // constructor.
            var client = new AmazonCloudWatchLogsClient();

            string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
            string groupName = "cloudwatchlogs-example-loggroup";

            var request = new AssociateKmsKeyRequest
```

```
            {
                KmsKeyId = kmsKeyId,
                LogGroupName = groupName,
            };

            var response = await client.AssociateKmsKeyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully associated KMS key ID: {kmsKeyId}
with log group: {groupName}.");
            }
            else
            {
                Console.WriteLine("Could not make the association between:
{kmsKeyId} and {groupName}.");
            }
        }
    }
```

- Para API obtener más información, consulte AssociateKmsKeyla AWS SDK for .NET APIReferencia.

`CancelExportTask`

En el siguiente ejemplo de código, se muestra cómo usar `CancelExportTask`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

```
/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

- Para API obtener más información, consulte CancelExportTaskla AWS SDK for .NET APIReferencia.

`CreateExportTask`

En el siguiente ejemplo de código, se muestra cómo usar `CreateExportTask`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskName = "export-task-example";
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string destination = "doc-example-bucket";
        var fromTime = 1437584472382;
        var toTime = 1437584472833;

        var request = new CreateExportTaskRequest
        {
            From = fromTime,
            To = toTime,
            TaskName = taskName,
            LogGroupName = logGroupName,
            Destination = destination,
        };
```

```
            var response = await client.CreateExportTaskAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"The task, {taskName} with ID: " +
                                  $"{response.TaskId} has been created
successfully.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateExportTaskla AWS SDK for .NET APIReferencia.

`CreateLogGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateLogGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to create an Amazon CloudWatch Logs log group.
/// </summary>
public class CreateLogGroup
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
```

```
            // as the default user on this system. If you need to use a
            // different AWS Region, pass it as a parameter to the client
            // constructor.
            var client = new AmazonCloudWatchLogsClient();

            string logGroupName = "cloudwatchlogs-example-loggroup";

            var request = new CreateLogGroupRequest
            {
                LogGroupName = logGroupName,
            };

            var response = await client.CreateLogGroupAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
            }
            else
            {
                Console.WriteLine("Could not create log group.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateLogGroupla AWS SDK for .NET APIReferencia.

`CreateLogStream`

En el siguiente ejemplo de código, se muestra cómo usar `CreateLogStream`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch
    /// log group.
    /// </summary>
    public class CreateLogStream
    {
        public static async Task Main()
        {
            // This client object will be associated with the same AWS Region
            // as the default user on this system. If you need to use a
            // different AWS Region, pass it as a parameter to the client
            // constructor.
            var client = new AmazonCloudWatchLogsClient();
            string logGroupName = "cloudwatchlogs-example-loggroup";
            string logStreamName = "cloudwatchlogs-example-logstream";

            var request = new CreateLogStreamRequest
            {
                LogGroupName = logGroupName,
                LogStreamName = logStreamName,
            };

            var response = await client.CreateLogStreamAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"{logStreamName} successfully created for
{logGroupName}.");
            }
            else
            {
                Console.WriteLine("Could not create stream.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateLogStreamla AWS SDK for .NET APIReferencia.

`DeleteLogGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteLogGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the Amazon CloudWatch Logs Service to delete an existing
    /// CloudWatch Logs log group.
    /// </summary>
    public class DeleteLogGroup
    {
        public static async Task Main()
        {
            var client = new AmazonCloudWatchLogsClient();
            string logGroupName = "cloudwatchlogs-example-loggroup";

            var request = new DeleteLogGroupRequest
            {
                LogGroupName = logGroupName,
            };

            var response = await client.DeleteLogGroupAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
```

```
            }
        }
    }
```

- Para API obtener más información, consulte DeleteLogGroupla AWS SDK for .NET APIReferencia.

`DescribeExportTasks`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeExportTasks`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to retrieve a list of information about Amazon CloudWatch
/// Logs export tasks.
/// </summary>
public class DescribeExportTasks
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();

        var request = new DescribeExportTasksRequest
```

```
            {
                Limit = 5,
            };

            var response = new DescribeExportTasksResponse();

            do
            {
                response = await client.DescribeExportTasksAsync(request);
                response.ExportTasks.ForEach(t =>
                {
                    Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has status:
{t.Status}");
                });
            }
            while (response.NextToken is not null);
        }
    }
```

- Para API obtener más información, consulte DescribeExportTasksla AWS SDK for .NET APIReferencia.

`DescribeLogGroups`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeLogGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

```
    /// <summary>
    /// Retrieves information about existing Amazon CloudWatch Logs log groups
    /// and displays the information on the console.
    /// </summary>
    public class DescribeLogGroups
    {
        public static async Task Main()
        {
            // Creates a CloudWatch Logs client using the default
            // user. If you need to work with resources in another
            // AWS Region than the one defined for the default user,
            // pass the AWS Region as a parameter to the client constructor.
            var client = new AmazonCloudWatchLogsClient();

            bool done = false;
            string newToken = null;

            var request = new DescribeLogGroupsRequest
            {
                Limit = 5,
            };

            DescribeLogGroupsResponse response;

            do
            {
                if (newToken is not null)
                {
                    request.NextToken = newToken;
                }

                response = await client.DescribeLogGroupsAsync(request);

                response.LogGroups.ForEach(lg =>
                {
                    Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
                    Console.WriteLine($"Created on: {lg.CreationTime.Date.Date}");
                    Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
                });

                if (response.NextToken is null)
                {
```

```
                    done = true;
                }
                else
                {
                    newToken = response.NextToken;
                }
            }
            while (!done);
        }
    }
```

- Para API obtener más información, consulte DescribeLogGroupsla AWS SDK for .NET APIReferencia.

`StartLiveTail`

En el siguiente ejemplo de código, se muestra cómo usar `StartLiveTail`.

AWS SDK for .NET

Incluir los archivos requeridos.

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Inicie la sesión de Live Tail.

```
            var client = new AmazonCloudWatchLogsClient();
            var request = new StartLiveTailRequest
            {
                LogGroupIdentifiers = logGroupIdentifiers,
                LogStreamNames = logStreamNames,
                LogEventFilterPattern = filterPattern,
            };

            var response = await client.StartLiveTailAsync(request);

            // Catch if request fails
```

```
            if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine("Failed to start live tail session");
                return;
            }
```

Puede controlar los eventos de la sesión de Live Tail de dos maneras:

```
            /* Method 1
            * 1). Asynchronously loop through the event stream
            * 2). Set a timer to dispose the stream and stop the Live Tail session
at the end.
            */
            var eventStream = response.ResponseStream;
            var task = Task.Run(() =>
            {
                foreach (var item in eventStream)
                {
                    if (item is LiveTailSessionUpdate liveTailSessionUpdate)
                    {
                        foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
                        {
                            Console.WriteLine("Message : {0}",
sessionResult.Message);
                        }
                    }
                    if (item is LiveTailSessionStart)
                    {
                        Console.WriteLine("Live Tail session started");
                    }
                    // On-stream exceptions are processed here
                    if (item is CloudWatchLogsEventStreamException)
                    {
                        Console.WriteLine($"ERROR: {item}");
                    }
                }
            });
            // Close the stream to stop the session after a timeout
            if (!task.Wait(TimeSpan.FromSeconds(10))){
                eventStream.Dispose();
                Console.WriteLine("End of line");
```

```
            }
```

```
            /* Method 2
            * 1). Add event handlers to each event variable
            * 2). Start processing the stream and wait for a timeout using
AutoResetEvent
            */
            AutoResetEvent endEvent = new AutoResetEvent(false);
            var eventStream = response.ResponseStream;
            using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
            {
                eventStream.SessionStartReceived += (sender, e) =>
                {
                    Console.WriteLine("LiveTail session started");
                };
                eventStream.SessionUpdateReceived += (sender, e) =>
                {
                    foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
                        Console.WriteLine("Message: {0}", logEvent.Message);
                    }
                };
                // On-stream exceptions are captured here
                eventStream.ExceptionReceived += (sender, e) =>
                {
                    Console.WriteLine($"ERROR: {e.EventStreamException.Message}");
                };

                eventStream.StartProcessing();
                // Stream events for this amount of time.
                endEvent.WaitOne(TimeSpan.FromSeconds(10));
                Console.WriteLine("End of line");
            }
```

- Para API obtener más información, consulte StartLiveTailen AWS SDK for .NET APIla Referencia.

Ejemplos de Amazon Cognito Identity Provider que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el AWS SDK for .NET uso de 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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`AdminGetUser`

En el siguiente ejemplo de código, se muestra cómo usar `AdminGetUser`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with administrator
access.
    /// </summary>
```

```
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }
```

- Para API obtener más información, consulte AdminGetUserla AWS SDK for .NET APIReferencia.

`AdminInitiateAuth`

En el siguiente ejemplo de código, se muestra cómo usar `AdminInitiateAuth`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
```

```
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }
```

- Para API obtener más información, consulte AdminInitiateAuthla AWS SDK for .NET APIReferencia.

`AdminRespondToAuthChallenge`

En el siguiente ejemplo de código, se muestra cómo usar `AdminRespondToAuthChallenge`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Respond to an admin authentication challenge.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
```

```
    /// <param name="clientId">The client ID.</param>
    /// <param name="mfaCode">The multi-factor authentication code.</param>
    /// <param name="session">The current application session.</param>
    /// <param name="clientId">The user pool ID.</param>
    /// <returns>The result of the authentication response.</returns>
    public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
        string userName,
        string clientId,
        string mfaCode,
        string session,
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };

        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }
```

- Para API obtener más información, consulte AdminRespondToAuthChallengela AWS SDK for .NET APIReferencia.

`AssociateSoftwareToken`

En el siguiente ejemplo de código, se muestra cómo usar `AssociateSoftwareToken`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }
```

- Para API obtener más información, consulte AssociateSoftwareTokenla AWS SDK for .NET APIReferencia.

`ConfirmDevice`

En el siguiente ejemplo de código, se muestra cómo usar `ConfirmDevice`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }
```

- Para API obtener más información, consulte ConfirmDevicela AWS SDK for .NET APIReferencia.

`ConfirmSignUp`

En el siguiente ejemplo de código, se muestra cómo usar `ConfirmSignUp`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            Console.WriteLine($"{userName} was confirmed");
            return true;
        }
        return false;
    }
```

- Para API obtener más información, consulte ConfirmSignUpla AWS SDK for .NET APIReferencia.

`InitiateAuth`

En el siguiente ejemplo de código, se muestra cómo usar `InitiateAuth`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }
```

- Para API obtener más información, consulte InitiateAuthla AWS SDK for .NET APIReferencia.

`ListUserPools`

En el siguiente ejemplo de código, se muestra cómo usar `ListUserPools`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 API obtener más información, consulte ListUserPoolsla AWS SDK for .NET APIReferencia.

`ListUsers`

En el siguiente ejemplo de código, se muestra cómo usar `ListUsers`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 users for the Amazon Cognito user pool.
/// </summary>
/// <param name="userPoolId">The user pool ID.</param>
/// <returns>A list of users.</returns>
public async Task<List<UserType>> ListUsersAsync(string userPoolId)
{
    var request = new ListUsersRequest
    {
        UserPoolId = userPoolId
    };

    var users = new List<UserType>();

    var usersPaginator = _cognitoService.Paginators.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- Para API obtener más información, consulte ListUsersla AWS SDK for .NET APIReferencia.

`ResendConfirmationCode`

En el siguiente ejemplo de código, se muestra cómo usar `ResendConfirmationCode`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);

        Console.WriteLine($"Method of delivery is
 {response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }
```

- Para API obtener más información, consulte ResendConfirmationCodela AWS SDK for .NET APIReferencia.

`SignUp`

En el siguiente ejemplo de código, se muestra cómo usar `SignUp`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte SignUpla AWS SDK for .NET APIReferencia.

`VerifySoftwareToken`

En el siguiente ejemplo de código, se muestra cómo usar `VerifySoftwareToken`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Verify the TOTP and register for MFA.
   /// </summary>
   /// <param name="session">The name of the session.</param>
   /// <param name="code">The MFA code.</param>
   /// <returns>The status of the software token.</returns>
   public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
   {
       var tokenRequest = new VerifySoftwareTokenRequest
       {
           UserCode = code,
           Session = session,
       };

       var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

       return verifyResponse.Status;
   }
```

- Para API obtener más información, consulte VerifySoftwareTokenla AWS SDK for .NET APIReferencia.

Escenarios

Registre un usuario con un grupo de usuarios que requiera MFA

En el siguiente ejemplo de código, se muestra cómo:

- Registre y confirme a un usuario con un nombre de usuario, una contraseña y una dirección de correo electrónico.
- Configure la autenticación multifactorial asociando una MFA aplicación al usuario.
- Inicie sesión con una contraseña y un MFA código.

AWS SDK for .NET

> **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 CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Cognito.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCognitoIdentityProvider>()
            .AddTransient<CognitoWrapper>()
            )
            .Build();
```

```
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CognitoBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

        Console.WriteLine(new string('-', 80));
        UiMethods.DisplayOverview();
        Console.WriteLine(new string('-', 80));

        // clientId - The app client Id value that you get from the AWS CDK script.
        var clientId = configuration["ClientId"]; // "*** REPLACE WITH CLIENT ID
VALUE FROM CDK SCRIPT";

        // poolId - The pool Id that you get from the AWS CDK script.
        var poolId = configuration["PoolId"]!; // "*** REPLACE WITH POOL ID VALUE
FROM CDK SCRIPT";
        var userName = configuration["UserName"];
        var password = configuration["Password"];
        var email = configuration["Email"];

        // If the username wasn't set in the configuration file,
        // get it from the user now.
        if (userName is null)
        {
            do
            {
                Console.Write("Username: ");
                userName = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(userName));
        }
        Console.WriteLine($"\nUsername: {userName}");

        // If the password wasn't set in the configuration file,
        // get it from the user now.
        if (password is null)
        {
```

```
            do
            {
                Console.Write("Password: ");
                password = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(password));
        }

        // If the email address wasn't set in the configuration file,
        // get it from the user now.
        if (email is null)
        {
            do
            {
                Console.Write("Email: ");
                email = Console.ReadLine();
            } while (string.IsNullOrEmpty(email));
        }

        // Now sign up the user.
        Console.WriteLine($"\nSigning up {userName} with email address: {email}");
        await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

        // Add the user to the user pool.
        Console.WriteLine($"Adding {userName} to the user pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        UiMethods.DisplayTitle("Get confirmation code");
        Console.WriteLine($"Conformation code sent to {userName}.");
        Console.Write("Would you like to send a new code? (Y/N) ");
        var answer = Console.ReadLine();

        if (answer!.ToLower() == "y")
        {
            await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
            Console.WriteLine("Sending a new confirmation code");
        }

        Console.Write("Enter confirmation code (from Email): ");
        var code = Console.ReadLine();

        await cognitoWrapper.ConfirmSignupAsync(clientId, code, userName);

        UiMethods.DisplayTitle("Checking status");
```

```
        Console.WriteLine($"Rechecking the status of {userName} in the user pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        Console.WriteLine($"Setting up authenticator for {userName} in the user
 pool");
        var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
 userName, password);

        var setupSession = await
 cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
        Console.Write("Enter the 6-digit code displayed in Google Authenticator: ");
        var setupCode = Console.ReadLine();

        var setupResult = await
 cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
        Console.WriteLine($"Setup status: {setupResult}");

        Console.WriteLine($"Now logging in {userName} in the user pool");
        var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
 poolId, userName, password);

        Console.Write("Enter a new 6-digit code displayed in Google Authenticator:
 ");
        var authCode = Console.ReadLine();

        var authResult = await
 cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
 authSession, poolId);
        Console.WriteLine($"Authenticated and received access token:
 {authResult.AccessToken}");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cognito scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
}


using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
```

```
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
    /// Constructor for the wrapper class containing Amazon Cognito actions.
    /// </summary>
    /// <param name="cognitoService">The Amazon Cognito client object.</param>
    public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)
    {
        _cognitoService = cognitoService;
    }

    /// <summary>
    /// List the Amazon Cognito user pools for an account.
    /// </summary>
    /// <returns>A list of UserPoolDescriptionType objects.</returns>
    public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
    {
        var userPools = new List<UserPoolDescriptionType>();

        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
 ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }


    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
```

```
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Respond to an admin authentication challenge.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="clientId">The client ID.</param>
    /// <param name="mfaCode">The multi-factor authentication code.</param>
    /// <param name="session">The current application session.</param>
    /// <param name="clientId">The user pool ID.</param>
    /// <returns>The result of the authentication response.</returns>
    public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
        string userName,
        string clientId,
        string mfaCode,
        string session,
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };
```

```
        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }


    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
    }


    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };
```

```
        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }


    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }

    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</param>
```

```
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }

    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
 userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
```

```
            Console.WriteLine($"{userName} was confirmed");
            return true;
        }
        return false;
    }


    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
 string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }


    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };
```

```
        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);

        Console.WriteLine($"Method of delivery is
 {response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }


    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with administrator
 access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
 poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }


    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
```

```
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers

- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp
- VerifySoftwareToken

Amazon Comprehend ejemplos utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET con Amazon Comprehend.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Escenarios

Acciones

`DetectDominantLanguage`

En el siguiente ejemplo de código, se muestra cómo usar `DetectDominantLanguage`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

   /// <summary>
   /// This example calls the Amazon Comprehend service to determine the
   /// dominant language.
   /// </summary>
   public static class DetectDominantLanguage
   {
       /// <summary>
       /// Calls Amazon Comprehend to determine the dominant language used in
       /// the sample text.
       /// </summary>
       public static async Task Main()
       {
           string text = "It is raining today in Seattle.";

           var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

           Console.WriteLine("Calling DetectDominantLanguage\n");
           var detectDominantLanguageRequest = new DetectDominantLanguageRequest()
           {
               Text = text,
           };

           var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
           foreach (var dl in detectDominantLanguageResponse.Languages)
           {
               Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
           }

           Console.WriteLine("Done");
       }
   }
```

- Para API obtener más información, consulte DetectDominantLanguagela AWS SDK for .NET APIReferencia.

`DetectEntities`

En el siguiente ejemplo de código, se muestra cómo usar `DetectEntities`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// This example shows how to use the AmazonComprehend service detect any
/// entities in submitted text.
/// </summary>
public static class DetectEntities
{
    /// <summary>
    /// The main method calls the DetectEntitiesAsync method to find any
    /// entities in the sample code.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new AmazonComprehendClient();

        Console.WriteLine("Calling DetectEntities\n");
        var detectEntitiesRequest = new DetectEntitiesRequest()
        {
            Text = text,
            LanguageCode = "en",
        };
```

```
            var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

            foreach (var e in detectEntitiesResponse.Entities)
            {
                Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- Para API obtener más información, consulte DetectEntitiesla AWS SDK for .NET APIReferencia.

`DetectKeyPhrases`

En el siguiente ejemplo de código, se muestra cómo usar `DetectKeyPhrases`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// This example shows how to use the Amazon Comprehend service to
    /// search text for key phrases.
    /// </summary>
    public static class DetectKeyPhrase
    {
```

```
        /// <summary>
        /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync
        /// to detect any key phrases in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
            foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
            {
                Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score}, BeginOffset:
{kp.BeginOffset}, EndOffset: {kp.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- Para API obtener más información, consulte DetectKeyPhrasesla AWS SDK for .NET APIReferencia.

`DetectPiiEntities`

En el siguiente ejemplo de código, se muestra cómo usar `DetectPiiEntities`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally dientifiable information within the supplied text.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();
        var text = @"Hello Paul Santos. The latest statement for your
                    credit card account 1111-0000-1111-0000 was
                    mailed to 123 Any Street, Seattle, WA 98109.";

        var request = new DetectPiiEntitiesRequest
        {
            Text = text,
            LanguageCode = "EN",
        };

        var response = await comprehendClient.DetectPiiEntitiesAsync(request);

        if (response.Entities.Count > 0)
        {
            foreach (var entity in response.Entities)
```

```
                {
                    var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                    Console.WriteLine($"{entity.Type}: {entityValue}");
                }
            }
        }
    }
```

- Para API obtener más información, consulte DetectPiiEntitiesla AWS SDK for .NET APIReferencia.

`DetectSentiment`

En el siguiente ejemplo de código, se muestra cómo usar `DetectSentiment`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// This example shows how to detect the overall sentiment of the supplied
/// text using the Amazon Comprehend service.
/// </summary>
public static class DetectSentiment
{
    /// <summary>
    /// This method calls the DetetectSentimentAsync method to analyze the
    /// supplied text and determine the overal sentiment.
    /// </summary>
```

```
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            var detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
            Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
            Console.WriteLine("Done");
        }
    }
```

- Para API obtener más información, consulte DetectSentimentla AWS SDK for .NET APIReferencia.

`DetectSyntax`

En el siguiente ejemplo de código, se muestra cómo usar `DetectSyntax`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Comprehend;
```

```
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use Amazon Comprehend to detect syntax
    /// elements by calling the DetectSyntaxAsync method.
    /// </summary>
    public class DetectingSyntax
    {
        /// <summary>
        /// This method calls DetectSynaxAsync to identify the syntax elements
        /// in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            // Call DetectSyntax API
            Console.WriteLine("Calling DetectSyntaxAsync\n");
            var detectSyntaxRequest = new DetectSyntaxRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            DetectSyntaxResponse detectSyntaxResponse = await
comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
            foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
            {
                Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
{s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- Para API obtener más información, consulte DetectSyntaxla AWS SDK for .NET APIReferencia.

`StartTopicsDetectionJob`

En el siguiente ejemplo de código, se muestra cómo usar `StartTopicsDetectionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// This example scans the documents in an Amazon Simple Storage Service
    /// (Amazon S3) bucket and analyzes it for topics. The results are stored
    /// in another bucket and then the resulting job properties are displayed
    /// on the screen. This example was created using the AWS SDK for .NEt
    /// version 3.7 and .NET Core version 5.0.
    /// </summary>
    public static class TopicModeling
    {
        /// <summary>
        /// This methos calls a topic detection job by calling the Amazon
        /// Comprehend StartTopicsDetectionJobRequest.
        /// </summary>
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();

            string inputS3Uri = "s3://input bucket/input path";
            InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
            string outputS3Uri = "s3://output bucket/output path";
            string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";
            int numberOfTopics = 10;

            var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
```

```
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat,
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri,
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics,
            };

            var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

            var jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);

            var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId,
            };

            var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);

PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
            foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
            {
                PrintJobProperties(props);
            }
        }

        /// <summary>
        /// This method is a helper method that displays the job properties
        /// from the call to StartTopicsDetectionJobRequest.
```

```
        /// </summary>
        /// <param name="props">A list of properties from the call to
        /// StartTopicsDetectionJobRequest.</param>
        private static void PrintJobProperties(TopicsDetectionJobProperties props)
        {
            Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
            Console.WriteLine($"NumberOfTopics: {props.NumberOfTopics}\nInputS3Uri:
{props.InputDataConfig.S3Uri}");
            Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
        }
    }
```

- Para API obtener más información, consulte StartTopicsDetectionJobla AWS SDK for .NET APIReferencia.

Escenarios

Creación de una aplicación para analizar los comentarios de los clientes

El siguiente ejemplo de código muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ejemplos de DynamoDB que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante DynamoDB. AWS SDK for .NET

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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a DynamoDB

En los siguientes ejemplos de código, se muestra cómo empezar a utilizar DynamoDB.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();

        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
 tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListTablesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones
- Escenarios
- Ejemplos sin servidor

Conceptos básicos

Aprenda los 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.
- Escriba los datos de la película en la tabla a partir de un JSON archivo de muestra.
- 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.

AWS SDK for .NET

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// This example application performs the following basic Amazon DynamoDB
// functions:
//
//     CreateTableAsync
//     PutItemAsync
//     UpdateItemAsync
//     BatchWriteItemAsync
```

```
//     GetItemAsync
//     DeleteItemAsync
//     Query
//     Scan
//     DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await DynamoDbMethods.CreateMovieTableAsync(client,
 tableName);

        if (success)
        {
            Console.WriteLine($"\nTable: {tableName} successfully created.");
        }
        else
        {
            Console.WriteLine($"\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 DynamoDbMethods.PutItemAsync(client, 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 DynamoDbMethods.UpdateItemAsync(client, 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 DynamoDbMethods.BatchWriteItemsAsync(client,
movieFileName);
        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 DynamoDbMethods.GetItemAsync(client, lookupMovie,
tableName);
        if (item.Count > 0)
        {
            DynamoDbMethods.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 DynamoDbMethods.DeleteItemAsync(client, 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($"Movies released in {findYear}");
        var queryCount = await DynamoDbMethods.QueryMoviesAsync(client, tableName,
findYear);
        Console.WriteLine($"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 DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await DynamoDbMethods.DeleteTableAsync(client, tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is done.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
```

```
    {
        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()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        Console.WriteLine(SepBar);
        _ = Console.ReadLine();
    }
}
```

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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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,
                    },
                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
```

```
                    WriteCapacityUnits = 5,
                },
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
 client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

Agrega una sola película a la tabla.

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing informtation 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Actualiza un solo elemento de una tabla.

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
```

```
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Recupera un solo elemento de la tabla de películas.

```
        /// <summary>
        /// Gets information about an existing movie from the table.
```

```
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

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 full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
```

```
            return null;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        return allMovies.GetRange(0, 250);
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="client">The initialized DynamoDB client object.</param>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public static async Task<long> BatchWriteItemsAsync(
        AmazonDynamoDBClient client,
        string movieFileName)
    {
        var movies = ImportMovies(movieFileName);
        if (movies is null)
        {
            Console.WriteLine("Couldn't find the JSON file with movie data.");
            return 0;
        }

        var context = new DynamoDBContext(client);

        var movieBatch = context.CreateBatchWrite<Movie>();
        movieBatch.AddPutItems(movies);

        Console.WriteLine("Adding imported movies to the table.");
        await movieBatch.ExecuteAsync();

        return movies.Count;
```

```
    }
```

Elimina un solo elemento de la tabla.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="client">The initialized DynamoDB client object.</param>
    /// <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 static async Task<bool> DeleteItemAsync(
        AmazonDynamoDBClient client,
        string tableName,
        Movie movieToDelete)
    {
        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,
        };

        var response = await client.DeleteItemAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

Busca en la tabla películas que se hayan estrenado en un intervalo de años.

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
```

```
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

Elimina la tabla de películas.

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable

- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Acciones

`BatchExecuteStatement`

En el siguiente ejemplo de código, se muestra cómo usar `BatchExecuteStatement`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Use lotes de INSERT estados de cuenta para agregar elementos.

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;
```

```
            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
```

```
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

Use lotes de SELECT estados de cuenta para obtener elementos.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
```

```
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
```

```
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

Use lotes de UPDATE estados de cuenta para actualizar los elementos.

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {
```

```
            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Use lotes de DELETE estados de cuenta para eliminar elementos.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
```

```
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
```

```
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte BatchExecuteStatementla AWS SDK for .NET APIReferencia.

`BatchGetItem`

En el siguiente ejemplo de código, se muestra cómo usar `BatchGetItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient
 client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
```

```
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
```

```
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
```

```
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
 attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- Para API obtener más información, consulte BatchGetItemla AWS SDK for .NET APIReferencia.

`BatchWriteItem`

En el siguiente ejemplo de código, se muestra cómo usar `BatchWriteItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonSerializer.Deserialize<List<Movie>>(
                json,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

            // Now return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
```

```
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

- Para API obtener más información, consulte BatchWriteItemla AWS SDK for .NET APIReferencia.

`CreateTable`

En el siguiente ejemplo de código, se muestra cómo usar `CreateTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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,
                },
            },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5,
            },
        });

        // Wait until the table is ACTIVE and then report success.
        Console.Write("Waiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = response.TableDescription.TableName,
        };

        TableStatus status;

        int sleepDuration = 2000;

        do
        {
            System.Threading.Thread.Sleep(sleepDuration);

            var describeTableResponse = await
client.DescribeTableAsync(request);
            status = describeTableResponse.Table.TableStatus;

            Console.Write(".");
        }
        while (status != "ACTIVE");

        return status == TableStatus.ACTIVE;
    }
```

- Para API obtener más información, consulte CreateTablela AWS SDK for .NET APIReferencia.

`DeleteItem`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">The initialized DynamoDB client object.</param>
/// <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 static async Task<bool> DeleteItemAsync(
    AmazonDynamoDBClient client,
    string tableName,
    Movie movieToDelete)
{
    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,
    };
```

```
            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte DeleteItemla AWS SDK for .NET APIReferencia.

`DeleteTable`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- Para API obtener más información, consulte DeleteTablela AWS SDK for .NET APIReferencia.

`DescribeTable`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    private static async Task GetTableInformation()
    {
        Console.WriteLine("\n*** Retrieving table information ***");

        var response = await Client.DescribeTableAsync(new DescribeTableRequest
        {
            TableName = ExampleTableName
        });

        var table = response.Table;
        Console.WriteLine($"Name: {table.TableName}");
        Console.WriteLine($"# of items: {table.ItemCount}");
        Console.WriteLine($"Provision Throughput (reads/sec): " +
                          $"{table.ProvisionedThroughput.ReadCapacityUnits}");
        Console.WriteLine($"Provision Throughput (writes/sec): " +
                          $"{table.ProvisionedThroughput.WriteCapacityUnits}");
    }
```

- Para API obtener más información, consulte DescribeTablela AWS SDK for .NET APIReferencia.

`ExecuteStatement`

En el siguiente ejemplo de código, se muestra cómo usar `ExecuteStatement`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Usa una INSERT declaración para añadir un artículo.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Usa una SELECT declaración para obtener un artículo.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Use una SELECT declaración para obtener una lista de artículos.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
```

```
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Usa una UPDATE declaración para actualizar un elemento.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
```

```
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Use una DELETE declaración para eliminar una sola película.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte ExecuteStatementla AWS SDK for .NET APIReferencia.

`GetItem`

En el siguiente ejemplo de código, se muestra cómo usar `GetItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

- Para API obtener más información, consulte GetItemla AWS SDK for .NET APIReferencia.

`ListTables`

En el siguiente ejemplo de código, se muestra cómo usar `ListTables`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    private static async Task ListMyTables()
    {
        Console.WriteLine("\n*** Listing tables ***");

        string lastTableNameEvaluated = null;
        do
        {
            var response = await Client.ListTablesAsync(new ListTablesRequest
            {
                Limit = 2,
                ExclusiveStartTableName = lastTableNameEvaluated
            });

            foreach (var name in response.TableNames)
            {
                Console.WriteLine(name);
            }

            lastTableNameEvaluated = response.LastEvaluatedTableName;
        } while (lastTableNameEvaluated != null);
    }
```

- Para API obtener más información, consulte ListTablesla AWS SDK for .NET APIReferencia.

`PutItem`

En el siguiente ejemplo de código, se muestra cómo usar `PutItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing informtation 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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
```

```
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte PutItemla AWS SDK for .NET APIReferencia.

`Query`

En el siguiente ejemplo de código, se muestra cómo usar `Query`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

- Para API obtener más información, consulte Query in AWS SDK for .NET APIReference.

Scan

En el siguiente ejemplo de código, se muestra cómo usar `Scan`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
```

```
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

- Para API obtener más información, consulte Escanear en AWS SDK for .NET APIreferencia.

`UpdateItem`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateItem`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
```

```
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte UpdateItemla AWS SDK for .NET APIReferencia.

Escenarios

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- APIPuerta de enlace
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una tabla de Amazon DynamoDB y utilice Amazon Simple Email Service (SESAmazon) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizar Amazon DynamoDB. NETAPIpara crear una aplicación web dinámica que realice un seguimiento de los datos de trabajo de DynamoDB.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. GitHub

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Consultar una tabla mediante lotes de instrucciones PartiQL

En el siguiente ejemplo de código, se muestra cómo:

- Obtenga un lote de artículos ejecutando múltiples SELECT estados de cuenta.
- Agregue un lote de elementos ejecutando varias INSERT declaraciones.
- Actualice un lote de elementos ejecutando varias UPDATE declaraciones.
- Elimine un lote de elementos mediante la ejecución de varias DELETE declaraciones.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = "moviedata.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();
```

```
// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
```

```
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1, year1,
 producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
```

```
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting the
 table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
```

```
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
```

```
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
```

```
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
```

```
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {
```

```
            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
```

```
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
        }
```

- Para API obtener más información, consulte BatchExecuteStatementla AWS SDK for .NET APIReferencia.

Consultar una tabla con PartiQL

En el siguiente ejemplo de código, se muestra cómo:

- Obtenga un artículo ejecutando una SELECT declaración.
- Agregue un elemento mediante la ejecución de una INSERT declaración.
- Actualice un elemento mediante la ejecución de una UPDATE declaración.
- Elimine un elemento mediante la ejecución de una DELETE declaración.

AWS SDK for .NET

> **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 PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
 AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
```

```
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully added.
```

```
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
```

```
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
```

```
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
```

```
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
 = ?";

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string, AttributeValue>>
 items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}
```

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte ExecuteStatementla AWS SDK for .NET APIReferencia.

Utilizar un modelo de documento

El siguiente ejemplo de código muestra cómo realizar operaciones de creación, lectura, actualización y eliminación (CRUD) y por lotes mediante un modelo de documento para DynamoDB y un. AWS SDK

Para obtener información, consulte Modelo de documento.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Realice CRUD operaciones utilizando un modelo de documento.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }
```

```
        /// <summary>
        /// Loads the contents of a DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to load.</param>
        /// <returns>A DynamoDB table object.</returns>
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
```

```
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
```

```
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
```

```
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await productCatalog.DeleteItemAsync(sampleBookId,
config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
```

```
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

Realice operaciones de escritura por lotes con un modelo de documento.

```
    /// <summary>
```

```
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown at
this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }
```

```
        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

Examine una tabla con un modelo de documento.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
    /// table for values.
    /// </summary>
    public class MidLevelScanOnly
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");

            await FindProductsWithNegativePrice(productCatalogTable);
            await FindProductsWithNegativePriceWithConfig(productCatalogTable);
        }

        /// <summary>
        /// Retrieves any products that have a negative price in a DynamoDB table.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePrice(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePrice:
printing .............");

                foreach (var document in documentList)
                {
```

```
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
```

```
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Consulte y examine una tabla con un modelo de documento.

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
```

```
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
```

```
          Table table)
       {
           DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
           var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
           filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

           // Use Query overloads that take the minimum required query parameters.
           Search search = table.Query(filter);

           do
           {
               var documentSet = await search.GetNextSetAsync();
               Console.WriteLine("\nFindRepliesInLast15Days:
printing ............");

               foreach (var document in documentSet)
               {
                   PrintDocument(document);
               }
           }
           while (!search.IsDone);
       }

       /// <summary>
       /// Retrieve replies made during a specific time period.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadSubject">The subject of the thread, which we are
       /// searching for replies.</param>
       public static async Task FindRepliesPostedWithinTimePeriod(
         Table table,
         string forumName,
         string threadSubject)
       {
           DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
           DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0, 0));

           var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadSubject);
```

```
            filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
       {
         "Message",
         "ReplyDateTime",
         "PostedBy",
       },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
       }

       /// <summary>
       /// Perform a query for replies made in the last 15 days using a DynamoDB
       /// QueryOperationConfig object.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadName">The bane of the thread that we are searching
       /// for replies.</param>
       public static async Task FindRepliesInLast15DaysWithConfig(
         Table table,
```

```
          string forumName,
          string threadName)
       {
           DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
           var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadName);
           filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

           var config = new QueryOperationConfig()
           {
               Filter = filter,

               // Optional parameters.
               Select = SelectValues.SpecificAttributes,
               AttributesToGet = new List<string>
               {
                 "Message",
                 "ReplyDateTime",
                 "PostedBy",
               },
               ConsistentRead = true,
           };

           Search search = table.Query(config);

           do
           {
               var documentSet = await search.GetNextSetAsync();
               Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .....");

               foreach (var document in documentSet)
               {
                   PrintDocument(document);
               }
           }
           while (!search.IsDone);
       }

       /// <summary>
       /// Displays the contents of the passed DynamoDB document on the console.
       /// </summary>
       /// <param name="document">A DynamoDB document to display.</param>
```

```
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Utilizar un modelo de persistencia de objetos de alto nivel

El siguiente ejemplo de código muestra cómo realizar operaciones de creación, lectura, actualización y eliminación (CRUD) y por lotes mediante un modelo de persistencia de objetos para DynamoDB y un. AWS SDK

Para obtener información, consulte Modelo de persistencia de objetos.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Realice CRUD operaciones utilizando un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class HighLevelItemCrud
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await PerformCRUDOperations(context);
        }

        public static async Task PerformCRUDOperations(IDynamoDBContext context)
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
```

```
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

Realice operaciones de escritura por lotes mediante un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Performs high-level batch write operations to an Amazon DynamoDB table.
    /// This example was written using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class HighLevelBatchWriteItem
    {
        public static async Task SingleTableBatchWrite(IDynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                Isbn = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
```

```
            ProductCategory = "Book",
            Title = "My book3 in batch write",
        };

        Book book2 = new Book
        {
            Id = 903,
            InPublication = true,
            Isbn = "903-11-11-1111",
            PageCount = "200",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book4 in batch write",
        };

        var bookBatch = context.CreateBatchWrite<Book>();
        bookBatch.AddPutItems(new List<Book> { book1, book2 });

        Console.WriteLine("Adding two books to ProductCatalog table.");
        await bookBatch.ExecuteAsync();
    }

    public static async Task MultiTableBatchWrite(IDynamoDBContext context)
    {
        // New Forum item.
        Forum newForum = new Forum
        {
            Name = "Test BatchWrite Forum",
            Threads = 0,
        };
        var forumBatch = context.CreateBatchWrite<Forum>();
        forumBatch.AddPutItem(newForum);

        // New Thread item.
        Thread newThread = new Thread
        {
            ForumName = "S3 forum",
            Subject = "My sample question",
            KeywordTags = new List<string> { "S3", "Bucket" },
            Message = "Message text",
        };

        DynamoDBOperationConfig config = new DynamoDBOperationConfig();
        config.SkipVersionCheck = true;
```

```
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

Mapee datos arbitrarios a una tabla mediante un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table, retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
```

```
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

Consulte y examine una tabla mediante un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
 threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn} \n
 No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15 days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the query.</
param>
        /// <param name="forumName">The name of the forum that we're interested
 in.</param>
```

```
        /// <param name="threadSubject">The thread object containing the query
 parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
 times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId, cfg);
            IEnumerable<Reply> latestReplies = await response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

 Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the query.</
param>
        /// <param name="forumName">The name of the forum that we're interested
 in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
```

```
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId, cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
```

```
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

Ejemplos sin servidor

Invocación de una función de Lambda desde un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una transmisión de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de DynamoDB con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una transmisión de DynamoDB. La función informa

los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
```

```
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

EC2Ejemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonEC2.

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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola Amazon EC2

Los siguientes ejemplos de código muestran cómo empezar a utilizar AmazonEC2.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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>A Task object.</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>();

        var request = new DescribeSecurityGroupsRequest
        {
            MaxResults = 10,
        };
```

```
        // Retrieve information about up to 10 Amazon EC2 security groups.
        var response = await ec2Client.DescribeSecurityGroupsAsync(request);

        // Now print the security groups returned by the call to
        // DescribeSecurityGroupsAsync.
        Console.WriteLine("Security Groups:");
        response.SecurityGroups.ForEach(group =>
        {
            Console.WriteLine($"Security group: {group.GroupName} ID:
 {group.GroupId}");
        });
    }
}
```

- Para API obtener más información, consulte DescribeSecurityGroupsla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones
- Escenarios

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Cree un par de claves y un grupo de seguridad.
- Selecciona una Amazon Machine Image (AMI) y un tipo de instancia compatible y, a continuación, crea una instancia.
- Detenga y vuelva a iniciar la instancia.
- Asocie una dirección IP elástica a su instancia.
- Conéctate a tu instancia con los recursos SSH y, a continuación, límpialos.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecute un escenario en un símbolo del sistema.

```
/// <summary>
/// Show Amazon Elastic Compute Cloud (Amazon EC2) Basics actions.
/// </summary>
public class EC2Basics
{
    /// <summary>
    /// Perform the actions defined for the Amazon EC2 Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 and Amazon Simple Systems
        // Management Service.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddTransient<EC2Wrapper>()
                    .AddTransient<SsmWrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();
        var ec2Methods = new EC2Wrapper(ec2Client);

        var ssmClient =
 host.Services.GetRequiredService<IAmazonSimpleSystemsManagement>();
        var ssmMethods = new SsmWrapper(ssmClient);
        var uiMethods = new UiMethods();
```

```
        var uniqueName = Guid.NewGuid().ToString();
        var keyPairName = "mvp-example-key-pair" + uniqueName;
        var groupName = "ec2-scenario-group" + uniqueName;
        var groupDescription = "A security group created for the EC2 Basics
scenario.";

        // Start the scenario.
        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the key pair.
        uiMethods.DisplayTitle("Create RSA key pair");
        Console.Write("Let's create an RSA key pair that you can be use to ");
        Console.WriteLine("securely connect to your EC2 instance.");
        var keyPair = await ec2Methods.CreateKeyPair(keyPairName);

        // Save key pair information to a temporary file.
        var tempFileName = ec2Methods.SaveKeyPair(keyPair);

        Console.WriteLine($"Created the key pair: {keyPair.KeyName} and saved it to:
{tempFileName}");
        string? answer;
        do
        {
            Console.Write("Would you like to list your existing key pairs? ");
            answer = Console.ReadLine();
        } while (answer!.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            // List existing key pairs.
            uiMethods.DisplayTitle("Existing key pairs");

            // Passing an empty string to the DescribeKeyPairs method will return
            // a list of all existing key pairs.
            var keyPairs = await ec2Methods.DescribeKeyPairs("");
            keyPairs.ForEach(kp =>
            {
                Console.WriteLine($"{kp.KeyName} created at: {kp.CreateTime}
Fingerprint: {kp.KeyFingerprint}");
            });
        }
        uiMethods.PressEnter();
```

```
        // Create the security group.
        Console.WriteLine("Let's create a security group to manage access to your
 instance.");
        var secGroupId = await ec2Methods.CreateSecurityGroup(groupName,
 groupDescription);
        Console.WriteLine("Let's add rules to allow all HTTP and HTTPS inbound
 traffic and to allow SSH only from your current IP address.");

        uiMethods.DisplayTitle("Security group information");
        var secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);

        Console.WriteLine($"Created security group {groupName} in your default
 VPC.");
        secGroups.ForEach(group =>
        {
            ec2Methods.DisplaySecurityGroupInfoAsync(group);
        });
        uiMethods.PressEnter();

        Console.WriteLine("Now we'll authorize the security group we just created so
 that it can");
        Console.WriteLine("access the EC2 instances you create.");
        var success = await ec2Methods.AuthorizeSecurityGroupIngress(groupName);

        secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);
        Console.WriteLine($"Now let's look at the permissions again.");
        secGroups.ForEach(group =>
        {
            ec2Methods.DisplaySecurityGroupInfoAsync(group);
        });
        uiMethods.PressEnter();

        // Get list of available Amazon Linux 2 Amazon Machine Images (AMIs).
        var parameters = await ssmMethods.GetParametersByPath("/aws/service/ami-
amazon-linux-latest");

        List<string> imageIds = parameters.Select(param => param.Value).ToList();

        var images = await ec2Methods.DescribeImages(imageIds);

        var i = 1;
        images.ForEach(image =>
        {
            Console.WriteLine($"\t{i++}\t{image.Description}");
```

```
        });

        int choice;
        bool validNumber = false;

        do
        {
            Console.Write("Please select an image: ");
            var selImage = Console.ReadLine();
            validNumber = int.TryParse(selImage, out choice);
        } while (!validNumber);

        var selectedImage = images[choice - 1];

        // Display available instance types.
        uiMethods.DisplayTitle("Instance Types");
        var instanceTypes = await
ec2Methods.DescribeInstanceTypes(selectedImage.Architecture);

        i = 1;
        instanceTypes.ForEach(instanceType =>
        {
            Console.WriteLine($"\t{i++}\t{instanceType.InstanceType}");
        });

        do
        {
            Console.Write("Please select an instance type: ");
            var selImage = Console.ReadLine();
            validNumber = int.TryParse(selImage, out choice);
        } while (!validNumber);

        var selectedInstanceType = instanceTypes[choice - 1].InstanceType;

        // Create an EC2 instance.
        uiMethods.DisplayTitle("Creating an EC2 Instance");
        var instanceId = await ec2Methods.RunInstances(selectedImage.ImageId,
selectedInstanceType, keyPairName, secGroupId);
        Console.Write("Waiting for the instance to start.");
        var isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Running);
```

```
        } while (!isRunning);

        uiMethods.PressEnter();

        var instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\nYou can use SSH to connect to your instance. For
 example:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        uiMethods.PressEnter();

        Console.WriteLine("Now we'll stop the instance and then start it again to
 see what's changed.");

        await ec2Methods.StopInstances(instanceId);
        var hasStopped = false;
        do
        {
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");

        Console.WriteLine("Now let's start it up again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);

        Console.WriteLine("\nLet's see what changed.");

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);
```

```
        Console.WriteLine("\nNotice the change in the SSH information:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        uiMethods.PressEnter();

        Console.WriteLine("Now we will stop the instance again. Then we will create
 and associate an");
        Console.WriteLine("Elastic IP address to use with our instance.");

        await ec2Methods.StopInstances(instanceId);
        hasStopped = false;
        do
        {
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Allocate Elastic IP address");
        Console.WriteLine("You can allocate an Elastic IP address and associate
 it with your instance\nto keep a consistent IP address even when your instance
 restarts.");
        var allocationId = await ec2Methods.AllocateAddress();
        Console.WriteLine("Now we will associate the Elastic IP address with our
 instance.");
        var associationId = await ec2Methods.AssociateAddress(allocationId,
 instanceId);

        // Start the instance again.
        Console.WriteLine("Now let's start the instance again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);
```

```
        Console.WriteLine("\nLet's see what changed.");

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("Instance information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\nHere is the SSH information:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        Console.WriteLine("Let's stop and start the instance again.");
        uiMethods.PressEnter();

        await ec2Methods.StopInstances(instanceId);

        hasStopped = false;
        do
        {
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");

        Console.WriteLine("Now let's start it up again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);
        Console.WriteLine("Note that the IP address did not change this time.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Clean up resources");

        Console.WriteLine("Now let's clean up the resources we created.");
```

```
        // Terminate the instance.
        Console.WriteLine("Terminating the instance we created.");
        var stateChange = await ec2Methods.TerminateInstances(instanceId);

        // Wait for the instance state to be terminated.
        var hasTerminated = false;
        do
        {
            hasTerminated = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Terminated);
        } while (!hasTerminated);

        Console.WriteLine($"\nThe instance {instanceId} has been terminated.");
        Console.WriteLine("Now we can disassociate the Elastic IP address and
 release it.");

        // Disassociate the Elastic IP address.
        var disassociated = ec2Methods.DisassociateIp(associationId);

        // Delete the Elastic IP address.
        var released = ec2Methods.ReleaseAddress(allocationId);

        // Delete the security group.
        Console.WriteLine($"Deleting the Security Group: {groupName}.");
        success = await ec2Methods.DeleteSecurityGroup(secGroupId);
        if (success)
        {
            Console.WriteLine($"Successfully deleted {groupName}.");
        }

        // Delete the RSA key pair.
        Console.WriteLine($"Deleting the key pair: {keyPairName}");
        await ec2Methods.DeleteKeyPair(keyPairName);
        Console.WriteLine("Deleting the temporary file with the key information.");
        ec2Methods.DeleteTempFile(tempFileName);
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("EC2 Basics Scenario completed.");
        uiMethods.PressEnter();
    }
}
```

Defina una clase que abarque EC2 las acciones.

```
/// <summary>
/// Methods of this class perform Amazon Elastic Compute Cloud (Amazon EC2).
/// </summary>
public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEC2;

    public EC2Wrapper(IAmazonEC2 amazonService)
    {
        _amazonEC2 = amazonService;
    }

    /// <summary>
    /// Allocate an Elastic IP address.
    /// </summary>
    /// <returns>The allocation Id of the allocated address.</returns>
    public async Task<string> AllocateAddress()
    {
        var request = new AllocateAddressRequest();

        var response = await _amazonEC2.AllocateAddressAsync(request);
        return response.AllocationId;
    }

    /// <summary>
    /// Associate an Elastic IP address to an EC2 instance.
    /// </summary>
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>
    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
 instanceId)
    {
        var request = new AssociateAddressRequest
        {
            AllocationId = allocationId,
            InstanceId = instanceId
        };
```

```
        var response = await _amazonEC2.AssociateAddressAsync(request);
        return response.AssociationId;
    }

    /// <summary>
    /// Authorize the local computer ingress to EC2 instances associated
    /// with the virtual private cloud (VPC) security group.
    /// </summary>
    /// <param name="groupName">The name of the security group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
    {
        // Get the IP address for the local computer.
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges = new List<IpRange> { new IpRange { CidrIp =
 $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Authorize the local computer for ingress to
    /// the Amazon EC2 SecurityGroup.
    /// </summary>
    /// <returns>The IPv4 address of the computer running the scenario.</returns>
    private static async Task<string> GetIpAddress()
    {
        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
```

```
        return ipString.Trim();
    }

    /// <summary>
    /// Create an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name for the new key pair.</param>
    /// <returns>The Amazon EC2 key pair created.</returns>
    public async Task<KeyPair?> CreateKeyPair(string keyPairName)
    {
        var request = new CreateKeyPairRequest
        {
            KeyName = keyPairName,
        };

        var response = await _amazonEC2.CreateKeyPairAsync(request);

        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            var kp = response.KeyPair;
            return kp;
        }
        else
        {
            Console.WriteLine("Could not create key pair.");
            return null;
        }
    }

    /// <summary>
    /// Save KeyPair information to a temporary file.
    /// </summary>
    /// <param name="keyPair">The name of the key pair.</param>
    /// <returns>The full path to the temporary file.</returns>
    public string SaveKeyPair(KeyPair keyPair)
    {
        var tempPath = Path.GetTempPath();
        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);
```

```
        return pemFileName;
    }

    /// <summary>
    /// Create an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>
    public async Task<string> CreateSecurityGroup(string groupName, string
 groupDescription)
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        return response.GroupId;
    }


    /// <summary>
    /// Create a new Amazon EC2 VPC.
    /// </summary>
    /// <param name="cidrBlock">The CIDR block for the new security group.</param>
    /// <returns>The VPC Id of the new VPC.</returns>
    public async Task<string?> CreateVPC(string cidrBlock)
    {

        try
        {
            var response = await _amazonEC2.CreateVpcAsync(new CreateVpcRequest
            {
                CidrBlock = cidrBlock,
            });

            Vpc vpc = response.Vpc;
            Console.WriteLine($"Created VPC with ID: {vpc.VpcId}.");
            return vpc.VpcId;
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"Couldn't create VPC because: {ex.Message}");
            return null;
```

```
        }
    }

    /// <summary>
    /// Delete an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyPair(string keyPairName)
    {
        try
        {
            await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete the temporary file where the key pair information was saved.
    /// </summary>
    /// <param name="tempFileName">The path to the temporary file.</param>
    public void DeleteTempFile(string tempFileName)
    {
        if (File.Exists(tempFileName))
        {
            File.Delete(tempFileName);
        }
    }

    /// <summary>
    /// Delete an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteSecurityGroup(string groupId)
    {
```

```
        var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an Amazon EC2 VPC.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteVpc(string vpcId)
    {
        var request = new DeleteVpcRequest
        {
            VpcId = vpcId,
        };

        var response = await _amazonEC2.DeleteVpcAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Get information about existing Amazon EC2 images.
    /// </summary>
    /// <returns>A list of image information.</returns>
    public async Task<List<Image>> DescribeImages(List<string>? imageIds)
    {
        var request = new DescribeImagesRequest();
        if (imageIds is not null)
        {
            // If the imageIds list is not null, add the list
            // to the request object.
            request.ImageIds = imageIds;
        }

        var response = await _amazonEC2.DescribeImagesAsync(request);
        return response.Images;
    }

    /// <summary>
    /// Display the information returned by DescribeImages.
    /// </summary>
    /// <param name="images">The list of image information to display.</param>
    public void DisplayImageInfo(List<Image> images)
```

```
    {
        images.ForEach(image =>
        {
            Console.WriteLine($"{image.Name} Created on: {image.CreationDate}");
        });

    }

    /// <summary>
    /// Get information about an Amazon EC2 instance.
    /// </summary>
    /// <param name="instanceId">The instance Id of the EC2 instance.</param>
    /// <returns>An EC2 instance.</returns>
    public async Task<Instance> DescribeInstance(string instanceId)
    {
        var response = await _amazonEC2.DescribeInstancesAsync(
            new DescribeInstancesRequest { InstanceIds = new List<string>
{ instanceId } });
        return response.Reservations[0].Instances[0];
    }

    /// <summary>
    /// Display EC2 instance information.
    /// </summary>
    /// <param name="instance">The instance Id of the EC2 instance.</param>
    public void DisplayInstanceInformation(Instance instance)
    {
        Console.WriteLine($"ID: {instance.InstanceId}");
        Console.WriteLine($"Image ID: {instance.ImageId}");
        Console.WriteLine($"{instance.InstanceType}");
        Console.WriteLine($"Key Name: {instance.KeyName}");
        Console.WriteLine($"VPC ID: {instance.VpcId}");
        Console.WriteLine($"Public IP: {instance.PublicIpAddress}");
        Console.WriteLine($"State: {instance.State.Name}");
    }

    /// <summary>
    /// Get information about existing EC2 images.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task DescribeInstances()
    {
        // List all EC2 instances.
        await GetInstanceDescriptions();
```

```
        string tagName = "IncludeInList";
        string tagValue = "Yes";
        await GetInstanceDescriptionsFiltered(tagName, tagValue);
    }

    /// <summary>
    /// Get information for all existing Amazon EC2 instances.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptions()
    {
        Console.WriteLine("Showing all instances:");
        var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId}");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }

    /// <summary>
    /// Get information about EC2 instances filtered by a tag name and value.
    /// </summary>
    /// <param name="tagName">The name of the tag to filter on.</param>
    /// <param name="tagValue">The value of the tag to look for.</param>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
    {
        // This tag filters the results of the instance list.
        var filters = new List<Filter>
        {
            new Filter
            {
                Name = $"tag:{tagName}",
```

```
                Values = new List<string>
                {
                    tagValue,
                },
            },
        };
        var request = new DescribeInstancesRequest
        {
            Filters = filters,
        };

        Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to
\"Yes\".");
        var paginator = _amazonEC2.Paginators.DescribeInstances(request);

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }

    /// <summary>
    /// Describe the instance types available.
    /// </summary>
    /// <returns>A list of instance type information.</returns>
    public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
            { new Filter("processor-info.supported-architecture", new List<string>
{ architecture.ToString() }) };
        filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();
```

```
        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
        {
            instanceTypes.Add(instanceType);
        }
        return instanceTypes;
    }

    /// <summary>
    /// Display the instance type information returned by
DescribeInstanceTypesAsync.
    /// </summary>
    /// <param name="instanceTypes">The list of instance type information.</param>
    public void DisplayInstanceTypeInfo(List<InstanceTypeInfo> instanceTypes)
    {
        instanceTypes.ForEach(type =>
        {
            Console.WriteLine($"{type.InstanceType}\t{type.MemoryInfo}");
        });
    }

    /// <summary>
    /// Get information about an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair.</param>
    /// <returns>A list of key pair information.</returns>
    public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
    {
        var request = new DescribeKeyPairsRequest();
        if (!string.IsNullOrEmpty(keyPairName))
        {
            request = new DescribeKeyPairsRequest
            {
                KeyNames = new List<string> { keyPairName }
            };
        }
        var response = await _amazonEC2.DescribeKeyPairsAsync(request);
        return response.KeyPairs.ToList();
    }


    /// <summary>
    /// Retrieve information for an Amazon EC2 security group.
```

```
    /// </summary>
    /// <param name="groupId">The Id of the Amazon EC2 security group.</param>
    /// <returns>A list of security group information.</returns>
    public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
    {
        var request = new DescribeSecurityGroupsRequest();
        var groupIds = new List<string> { groupId };
        request.GroupIds = groupIds;

        var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
        return response.SecurityGroups;
    }

    /// <summary>
    /// Display the information returned by the call to
    /// DescribeSecurityGroupsAsync.
    /// </summary>
    /// <param name="securityGroup">A list of security group information.</param>
    public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
    {
        Console.WriteLine($"{securityGroup.GroupName}");
        Console.WriteLine("Ingress permissions:");
        securityGroup.IpPermissions.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
        Console.WriteLine("Egress permissions:");
        securityGroup.IpPermissionsEgress.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
```

```
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
    }


    /// <summary>
    /// Disassociate an Elastic IP address from an EC2 instance.
    /// </summary>
    /// <param name="associationId">The association Id.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DisassociateIp(string associationId)
    {
        var response = await _amazonEC2.DisassociateAddressAsync(
            new DisassociateAddressRequest { AssociationId = associationId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Retrieve a list of available Amazon Linux images.
    /// </summary>
    /// <returns>A list of image information.</returns>
    public async Task<List<Image>> GetEC2AmiList()
    {
        var filter = new Filter { Name = "architecture", Values = new List<string>
{ "x86_64" } };
        var filters = new List<Filter> { filter };
        var response = await _amazonEC2.DescribeImagesAsync(new
DescribeImagesRequest { Filters = filters });
        return response.Images;
    }
```

```
    /// <summary>
    /// Reboot EC2 instances.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the instances that will be
 rebooted.</param>
    /// <returns>Async task.</returns>
    public async Task RebootInstances(string ec2InstanceId)
    {
        var request = new RebootInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.RebootInstancesAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Instances successfully rebooted.");
        }
        else
        {
            Console.WriteLine("Could not reboot one or more instances.");
        }
    }

    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Create and run an EC2 instance.
    /// </summary>
```

```
    /// <param name="ImageId">The image Id of the image used as a basis for the
    /// EC2 instance.</param>
    /// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
    /// <param name="keyName">The name of the key pair to associate with the
    /// instance.</param>
    /// <param name="groupId">The Id of the Amazon EC2 security group that will be
    /// allowed to interact with the new EC2 instance.</param>
    /// <returns>The instance Id of the new EC2 instance.</returns>
    public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
    {
        var request = new RunInstancesRequest
        {
            ImageId = imageId,
            InstanceType = instanceType,
            KeyName = keyName,
            MinCount = 1,
            MaxCount = 1,
            SecurityGroupIds = new List<string> { groupId }
        };
        var response = await _amazonEC2.RunInstancesAsync(request);
        return response.Reservation.Instances[0].InstanceId;
    }


    /// <summary>
    /// Start an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
    /// to start.</param>
    /// <returns>Async task.</returns>
    public async Task StartInstances(string ec2InstanceId)
    {
        var request = new StartInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StartInstancesAsync(request);

        if (response.StartingInstances.Count > 0)
        {
            var instances = response.StartingInstances;
```

```
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
            });
        }
    }

    /// <summary>
    /// Stop an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance to
    /// stop.</param>
    /// <returns>Async task.</returns>
    public async Task StopInstances(string ec2InstanceId)
    {
        // In addition to the list of instance Ids, the
        // request can also include the following properties:
        //     Force      When true, forces the instances to
        //                stop but you must check the integrity
        //                of the file system. Not recommended on
        //                Windows instances.
        //     Hibernate  When true, hibernates the instance if the
        //                instance was enabled for hibernation when
        //                it was launched.
        var request = new StopInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StopInstancesAsync(request);

        if (response.StoppingInstances.Count > 0)
        {
            var instances = response.StoppingInstances;
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully stopped the EC2 Instance " +
                                  $"with InstanceID: {i.InstanceId}.");
            });
        }
    }

    /// <summary>
```

```
    /// Terminate an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance
    /// to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
    {
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };

        var response = await _amazonEC2.TerminateInstancesAsync(request);
        return response.TerminatingInstances;
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is running.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEC2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.Write(". ");
        } while (!hasState);
```

```
        return hasState;
    }

}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - AllocateAddress
  - AssociateAddress
  - AuthorizeSecurityGroupIngress
  - CreateKeyPair
  - CreateSecurityGroup
  - DeleteKeyPair
  - DeleteSecurityGroup
  - DescribeImages
  - DescribeInstanceTypes
  - DescribeInstances
  - DescribeKeyPairs
  - DescribeSecurityGroups
  - DisassociateAddress
  - ReleaseAddress
  - RunInstances
  - StartInstances
  - StopInstances
  - TerminateInstances
  - UnmonitorInstances

Acciones

`AllocateAddress`

En el siguiente ejemplo de código, se muestra cómo usar `AllocateAddress`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Allocate an Elastic IP address.
/// </summary>
/// <returns>The allocation Id of the allocated address.</returns>
public async Task<string> AllocateAddress()
{
    var request = new AllocateAddressRequest();

    var response = await _amazonEC2.AllocateAddressAsync(request);
    return response.AllocationId;
}
```

- Para API obtener más información, consulte AllocateAddressla AWS SDK for .NET APIReferencia.

`AssociateAddress`

En el siguiente ejemplo de código, se muestra cómo usar `AssociateAddress`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Associate an Elastic IP address to an EC2 instance.
/// </summary>
```

```
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>
    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
 instanceId)
    {
        var request = new AssociateAddressRequest
        {
            AllocationId = allocationId,
            InstanceId = instanceId
        };

        var response = await _amazonEC2.AssociateAddressAsync(request);
        return response.AssociationId;
    }
```

- Para API obtener más información, consulte AssociateAddressla AWS SDK for .NET APIReferencia.

`AuthorizeSecurityGroupIngress`

En el siguiente ejemplo de código, se muestra cómo usar `AuthorizeSecurityGroupIngress`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Authorize the local computer ingress to EC2 instances associated
    /// with the virtual private cloud (VPC) security group.
    /// </summary>
    /// <param name="groupName">The name of the security group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
    {
        // Get the IP address for the local computer.
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges = new List<IpRange> { new IpRange { CidrIp =
 $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Authorize the local computer for ingress to
    /// the Amazon EC2 SecurityGroup.
    /// </summary>
    /// <returns>The IPv4 address of the computer running the scenario.</returns>
    private static async Task<string> GetIpAddress()
    {
        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }
```

- Para API obtener más información, consulte AuthorizeSecurityGroupIngressla AWS SDK for .NET APIReferencia.

`CreateKeyPair`

En el siguiente ejemplo de código, se muestra cómo usar `CreateKeyPair`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Create an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name for the new key pair.</param>
/// <returns>The Amazon EC2 key pair created.</returns>
public async Task<KeyPair?> CreateKeyPair(string keyPairName)
{
    var request = new CreateKeyPairRequest
    {
        KeyName = keyPairName,
    };

    var response = await _amazonEC2.CreateKeyPairAsync(request);

    if (response.HttpStatusCode == HttpStatusCode.OK)
    {
        var kp = response.KeyPair;
        return kp;
    }
    else
    {
        Console.WriteLine("Could not create key pair.");
        return null;
    }
}

/// <summary>
/// Save KeyPair information to a temporary file.
/// </summary>
/// <param name="keyPair">The name of the key pair.</param>
/// <returns>The full path to the temporary file.</returns>
```

```
    public string SaveKeyPair(KeyPair keyPair)
    {
        var tempPath = Path.GetTempPath();
        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);

        return pemFileName;
    }
```

- Para API obtener más información, consulte CreateKeyPairla AWS SDK for .NET APIReferencia.

`CreateLaunchTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `CreateLaunchTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
```

```
   /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       await CreateKeyPair(_keyPairName);
       await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

       var startServerText = await File.ReadAllTextAsync(startupScriptPath);
       var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

       var amiLatest = await _amazonSsm.GetParameterAsync(
           new GetParameterRequest() { Name = _amiParam });
       var amiId = amiLatest.Parameter.Value;
       var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
           new CreateLaunchTemplateRequest()
           {
               LaunchTemplateName = _launchTemplateName,
               LaunchTemplateData = new RequestLaunchTemplateData()
               {
                   InstanceType = _instanceType,
                   ImageId = amiId,
                   IamInstanceProfile =
                       new
                           LaunchTemplateIamInstanceProfileSpecificationRequest()
                       {
                           Name = _instanceProfileName
                       },
                   KeyName = _keyPairName,
                   UserData = System.Convert.ToBase64String(plainTextBytes)
               }
           });
       return launchTemplateResponse.LaunchTemplate;

   }
```

- Para API obtener más información, consulte CreateLaunchTemplatela AWS SDK for .NET APIReferencia.

`CreateSecurityGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateSecurityGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>
    public async Task<string> CreateSecurityGroup(string groupName, string
 groupDescription)
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        return response.GroupId;
    }
```

- Para API obtener más información, consulte CreateSecurityGroupla AWS SDK for .NET APIReferencia.

`DeleteKeyPair`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteKeyPair`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyPair(string keyPairName)
    {
        try
        {
            await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete the temporary file where the key pair information was saved.
    /// </summary>
    /// <param name="tempFileName">The path to the temporary file.</param>
    public void DeleteTempFile(string tempFileName)
    {
        if (File.Exists(tempFileName))
        {
            File.Delete(tempFileName);
        }
    }
```

- Para API obtener más información, consulte DeleteKeyPairla AWS SDK for .NET APIReferencia.

`DeleteLaunchTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteLaunchTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Delete a launch template by name.
/// </summary>
/// <param name="templateName">The name of the template to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTemplateByName(string templateName)
{
    try
    {
        await _amazonEc2.DeleteLaunchTemplateAsync(
            new DeleteLaunchTemplateRequest()
            {
                LaunchTemplateName = templateName
            });
    }
    catch (AmazonClientException)
    {
        Console.WriteLine($"Unable to delete template {templateName}.");
    }
}
```

- Para API obtener más información, consulte DeleteLaunchTemplatela AWS SDK for .NET APIReferencia.

`DeleteSecurityGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteSecurityGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteSecurityGroup(string groupId)
    {
        var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteSecurityGroupla AWS SDK for .NET APIReferencia.

`DescribeAvailabilityZones`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAvailabilityZones`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }
```

- Para API obtener más información, consulte DescribeAvailabilityZonesla AWS SDK for .NET APIReferencia.

`DescribeIamInstanceProfileAssociations`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeIamInstanceProfileAssociations`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
```

```
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }
```

- Para API obtener más información, consulte DescribeIamInstanceProfileAssociationsla AWS SDK for .NET APIReferencia.

`DescribeInstanceTypes`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeInstanceTypes`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Describe the instance types available.
    /// </summary>
    /// <returns>A list of instance type information.</returns>
    public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
            { new Filter("processor-info.supported-architecture", new List<string>
{ architecture.ToString() }) };
        filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();

        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
```

```
        {
            instanceTypes.Add(instanceType);
        }
        return instanceTypes;
    }
```

- Para API obtener más información, consulte DescribeInstanceTypesla AWS SDK for .NET APIReferencia.

`DescribeInstances`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about existing EC2 images.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task DescribeInstances()
    {
        // List all EC2 instances.
        await GetInstanceDescriptions();

        string tagName = "IncludeInList";
        string tagValue = "Yes";
        await GetInstanceDescriptionsFiltered(tagName, tagValue);
    }

    /// <summary>
    /// Get information for all existing Amazon EC2 instances.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptions()
```

```
    {
        Console.WriteLine("Showing all instances:");
        var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId}");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }

    /// <summary>
    /// Get information about EC2 instances filtered by a tag name and value.
    /// </summary>
    /// <param name="tagName">The name of the tag to filter on.</param>
    /// <param name="tagValue">The value of the tag to look for.</param>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
    {
        // This tag filters the results of the instance list.
        var filters = new List<Filter>
        {
            new Filter
            {
                Name = $"tag:{tagName}",
                Values = new List<string>
                {
                    tagValue,
                },
            },
        };
        var request = new DescribeInstancesRequest
        {
            Filters = filters,
        };
```

```
        Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to
\"Yes\".");
        var paginator = _amazonEC2.Paginators.DescribeInstances(request);

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }
```

- Para API obtener más información, consulte DescribeInstancesla AWS SDK for .NET APIReferencia.

`DescribeKeyPairs`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeKeyPairs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair.</param>
    /// <returns>A list of key pair information.</returns>
    public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
    {
        var request = new DescribeKeyPairsRequest();
```

```
        if (!string.IsNullOrEmpty(keyPairName))
        {
            request = new DescribeKeyPairsRequest
            {
                KeyNames = new List<string> { keyPairName }
            };
        }
        var response = await _amazonEC2.DescribeKeyPairsAsync(request);
        return response.KeyPairs.ToList();
    }
```

- Para API obtener más información, consulte DescribeKeyPairsla AWS SDK for .NET APIReferencia.

`DescribeSecurityGroups`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeSecurityGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Retrieve information for an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupId">The Id of the Amazon EC2 security group.</param>
    /// <returns>A list of security group information.</returns>
    public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
    {
        var request = new DescribeSecurityGroupsRequest();
        var groupIds = new List<string> { groupId };
        request.GroupIds = groupIds;

        var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
        return response.SecurityGroups;
```

```
    }

    /// <summary>
    /// Display the information returned by the call to
    /// DescribeSecurityGroupsAsync.
    /// </summary>
    /// <param name="securityGroup">A list of security group information.</param>
    public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
    {
        Console.WriteLine($"{securityGroup.GroupName}");
        Console.WriteLine("Ingress permissions:");
        securityGroup.IpPermissions.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
        Console.WriteLine("Egress permissions:");
        securityGroup.IpPermissionsEgress.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
```

```
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
    }
```

- Para API obtener más información, consulte DescribeSecurityGroupsla AWS SDK for .NET APIReferencia.

`DescribeSubnets`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeSubnets`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
```

```
            }
        });

    // Get the entire list using the paginator.
    await foreach (var subnet in subnetPaginator.Subnets)
    {
        subnets.Add(subnet);
    }

    return subnets;
}
```

- Para API obtener más información, consulte DescribeSubnetsla AWS SDK for .NET APIReferencia.

`DescribeVpcs`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeVpcs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
```

```
            });
        return vpcResponse.Vpcs[0];
    }
```

- Para API obtener más información, consulte DescribeVpcsla AWS SDK for .NET APIReferencia.

`DisassociateAddress`

En el siguiente ejemplo de código, se muestra cómo usar `DisassociateAddress`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    var response = await _amazonEC2.DisassociateAddressAsync(
        new DisassociateAddressRequest { AssociationId = associationId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte DisassociateAddressla AWS SDK for .NET APIReferencia.

`RebootInstances`

En el siguiente ejemplo de código, se muestra cómo usar `RebootInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Reboot EC2 instances.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the instances that will be
rebooted.</param>
    /// <returns>Async task.</returns>
    public async Task RebootInstances(string ec2InstanceId)
    {
        var request = new RebootInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.RebootInstancesAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Instances successfully rebooted.");
        }
        else
        {
            Console.WriteLine("Could not reboot one or more instances.");
        }
    }
```

Sustituya el perfil por una instancia y reinicie un servidor web.

```
    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
```

```
   /// <param name="instanceId">The Id of the instance to update.</param>
   /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
   /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
                   AssociationId = associationId,
                   IamInstanceProfile = new IamInstanceProfileSpecification()
                   {
                       Name = credsProfileName
                   }
               });
       // Allow time before resetting.
       Thread.Sleep(25000);
       var instanceReady = false;
       var retries = 5;
       while (retries-- > 0 && !instanceReady)
       {
           await _amazonEc2.RebootInstancesAsync(
               new RebootInstancesRequest(new List<string>() { instanceId }));
           Thread.Sleep(10000);

           var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
               new DescribeInstanceInformationRequest());
           // Get the entire list using the paginator.
           await foreach (var instance in
instancesPaginator.InstanceInformationList)
           {
               instanceReady = instance.InstanceId == instanceId;
               if (instanceReady)
               {
                   break;
               }
           }
       }
       Console.WriteLine($"Sending restart command to instance {instanceId}");
       await _amazonSsm.SendCommandAsync(
```

```
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
 server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

- Para API obtener más información, consulte RebootInstancesla AWS SDK for .NET APIReferencia.

`ReleaseAddress`

En el siguiente ejemplo de código, se muestra cómo usar `ReleaseAddress`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };
```

```
        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte ReleaseAddressla AWS SDK for .NET APIReferencia.

`ReplaceIamInstanceProfileAssociation`

En el siguiente ejemplo de código, se muestra cómo usar `ReplaceIamInstanceProfileAssociation`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
```

```
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

- Para API obtener más información, consulte ReplaceIamInstanceProfileAssociationla AWS SDK for .NET APIReferencia.

`RunInstances`

En el siguiente ejemplo de código, se muestra cómo usar `RunInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create and run an EC2 instance.
   /// </summary>
   /// <param name="ImageId">The image Id of the image used as a basis for the
   /// EC2 instance.</param>
   /// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
   /// <param name="keyName">The name of the key pair to associate with the
   /// instance.</param>
   /// <param name="groupId">The Id of the Amazon EC2 security group that will be
   /// allowed to interact with the new EC2 instance.</param>
   /// <returns>The instance Id of the new EC2 instance.</returns>
   public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
   {
       var request = new RunInstancesRequest
       {
           ImageId = imageId,
           InstanceType = instanceType,
           KeyName = keyName,
           MinCount = 1,
           MaxCount = 1,
           SecurityGroupIds = new List<string> { groupId }
       };
```

```
        var response = await _amazonEC2.RunInstancesAsync(request);
        return response.Reservation.Instances[0].InstanceId;
    }
```

- Para API obtener más información, consulte RunInstancesla AWS SDK for .NET APIReferencia.

`StartInstances`

En el siguiente ejemplo de código, se muestra cómo usar `StartInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Start an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
    /// to start.</param>
    /// <returns>Async task.</returns>
    public async Task StartInstances(string ec2InstanceId)
    {
        var request = new StartInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StartInstancesAsync(request);

        if (response.StartingInstances.Count > 0)
        {
            var instances = response.StartingInstances;
            instances.ForEach(i =>
            {
```

```
                Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
            });
        }
    }
```

- Para API obtener más información, consulte StartInstancesla AWS SDK for .NET APIReferencia.

`StopInstances`

En el siguiente ejemplo de código, se muestra cómo usar `StopInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Stop an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance to
    /// stop.</param>
    /// <returns>Async task.</returns>
    public async Task StopInstances(string ec2InstanceId)
    {
        // In addition to the list of instance Ids, the
        // request can also include the following properties:
        //     Force      When true, forces the instances to
        //                stop but you must check the integrity
        //                of the file system. Not recommended on
        //                Windows instances.
        //     Hibernate  When true, hibernates the instance if the
        //                instance was enabled for hibernation when
        //                it was launched.
        var request = new StopInstancesRequest
        {
```

```
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StopInstancesAsync(request);

        if (response.StoppingInstances.Count > 0)
        {
            var instances = response.StoppingInstances;
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully stopped the EC2 Instance " +
                                  $"with InstanceID: {i.InstanceId}.");
            });
        }
    }
```

- Para API obtener más información, consulte StopInstancesla AWS SDK for .NET APIReferencia.

`TerminateInstances`

En el siguiente ejemplo de código, se muestra cómo usar `TerminateInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Terminate an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance
    /// to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
    {
```

```
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };

        var response = await _amazonEC2.TerminateInstancesAsync(request);
        return response.TerminatingInstances;
    }
```

- Para API obtener más información, consulte TerminateInstancesla AWS SDK for .NET APIReferencia.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (AmazonEC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias en un rango específico.
- Gestione y distribuya HTTP las solicitudes con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecuta un servidor web Python en cada EC2 instancia para gestionar HTTP las solicitudes. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

AWS SDK for .NET

> **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 el escenario interactivo en un símbolo del sistema.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
```

```
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
```

```
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
```

```
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
```

```
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);
```

```
        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
```

```
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
                }
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }
```

```
    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
 different parts of the system\n" +
                          "to create situations where the web service fails, and
 shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
 these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
 recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
 recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();
```

```
        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
```

```
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
```

```
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
```

```
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Crea una clase que agrupe las EC2 acciones de Auto Scaling y Amazon.

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
```

```
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {
```

```
        var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                     "\"Service\": [" +
                                         "\"ec2.amazonaws.com\"" +
                                     "]" +
                                     "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }
```

```
            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
```

```
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
```

```
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);
```

```
        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
```

```
    public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
```

```
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
```

```
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
```

```
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
```

```
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);
```

```
            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
```

```
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
```

```
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
```

```
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }
```

```
                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                      "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
```

```
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Cree una clase que resuma las acciones de Elastic Load Balancing.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
```

```
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }
```

```
    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
```

```
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
 subnets
    /// and forwards requests to the specified target group.
    /// </summary>
```

```
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });

               var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

               loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
           }
           catch (LoadBalancerNotFoundException)
           {
               loadBalancerReady = false;
           }
           Thread.Sleep(10000);
       }
       // Create the listener.
       await _amazonElasticLoadBalancingV2.CreateListenerAsync(
           new CreateListenerRequest()
```

```
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
```

```
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
 {groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

Cree una clase que utilice DynamoDB para simular un servicio de recomendaciones.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
```

```
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
```

```
                        AttributeName = "ItemId",
                        AttributeType = ScalarAttributeType.N
                    }
                },
            KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement()
                    {
                        AttributeName = "MediaType",
                        KeyType = KeyType.HASH
                    },
                    new KeySchemaElement()
                    {
                        AttributeName = "ItemId",
                        KeyType = KeyType.RANGE
                    }
                },
            ProvisionedThroughput = new ProvisionedThroughput()
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5
            }
        };
        await _amazonDynamoDb.CreateTableAsync(createRequest);

        // Wait until the table is ACTIVE and then report success.
        Console.Write("\nWaiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = tableName
        };

        TableStatus status;
        do
        {
            Thread.Sleep(2000);

            var describeTableResponse = await
_amazonDynamoDb.DescribeTableAsync(request);
            status = describeTableResponse.Table.TableStatus;

            Console.Write(".");
        }
```

```
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
```

```
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Cree una clase que agrupe las acciones de Systems Manager.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
```

```
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer

- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

ECSEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonECS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola Amazon ECS

El siguiente ejemplo de código muestra cómo empezar a utilizar AmazonECS.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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.Hosting;

namespace ECSActions;

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 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args).Build();

        // Now the client is available for injection.
        var amazonECSClient = new AmazonECSClient();

        // You can use await and any of the async methods to get a response.
        var response = await amazonECSClient.ListClustersAsync(new
 ListClustersRequest { });

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your aws account");
        Console.WriteLine();
        foreach (var arn in response.ClusterArns.Take(5))
        {
            Console.WriteLine($"\tARN: {arn}");
            Console.WriteLine($"Cluster Name: {arn.Split("/").Last()}");
```

```
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListClustersla AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios

Acciones

`ListClusters`

En el siguiente ejemplo de código, se muestra cómo usar `ListClusters`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List cluster ARNs available.
    /// </summary>
    /// <returns>The ARN list of clusters.</returns>
    public async Task<List<string>> GetClusterARNSAsync()
    {

        Console.WriteLine("Getting a list of all the clusters in your AWS
account...");
        List<string> clusterArnList = new List<string>();
        // Get a list of all the clusters in your AWS account
        try
        {
```

```
            var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
            {
            });

            var clusterArns = listClustersResponse.ClusterArns;

            // Print the ARNs of the clusters
            await foreach (var clusterArn in clusterArns)
            {
                clusterArnList.Add(clusterArn);
            }

            if (clusterArnList.Count == 0)
            {
                _logger.LogWarning("No clusters found in your AWS account.");
            }
            return clusterArnList;
        }
        catch (Exception e)
        {
            _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
            throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        }
    }
```

- Para API obtener más información, consulte ListClustersla AWS SDK for .NET APIReferencia.

`ListServices`

En el siguiente ejemplo de código, se muestra cómo usar `ListServices`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List service ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of services in given cluster.</returns>
    public async Task<List<string>> GetServiceARNSAsync(string clusterARN)
    {
        List<string> serviceArns = new List<string>();

        var request = new ListServicesRequest
        {
            Cluster = clusterARN
        };
        // Call the ListServices API operation and get the list of service ARNs
        var serviceList = _ecsClient.Paginators.ListServices(request);

        await foreach (var serviceARN in serviceList.ServiceArns)
        {
            if (serviceARN is null)
                continue;

            serviceArns.Add(serviceARN);
        }

        if (serviceArns.Count == 0)
        {
            _logger.LogWarning($"No services found in cluster {clusterARN} .");
        }

        return serviceArns;
    }
```

- Para API obtener más información, consulte ListServicesla AWS SDK for .NET APIReferencia.

`ListTasks`

En el siguiente ejemplo de código, se muestra cómo usar `ListTasks`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// List task ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of tasks in given cluster.</returns>
public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
{
    // Set up the request to describe the tasks in the service
    var listTasksRequest = new ListTasksRequest
    {
        Cluster = clusterARN
    };
    List<string> taskArns = new List<string>();

    // Call the ListTasks API operation and get the list of task ARNs
    var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

    await foreach (var task in tasks.TaskArns)
    {
        if (task is null)
            continue;


        taskArns.Add(task);
    }

    if (taskArns.Count == 0)
    {
        _logger.LogWarning("No tasks found in cluster: " + clusterARN);
    }

    return taskArns;
}
```

- Para API obtener más información, consulte ListTasksla AWS SDK for .NET APIReferencia.

Escenarios

Obtenga ARN información sobre clústeres, servicios y tareas

En el siguiente ejemplo de código, se muestra cómo:

- Obtener una lista de todos los clústeres.
- Obtener los servicios de un clúster.
- Obtener las tareas de un clúster.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto 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.ECS;
using ECSActions;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSScenario;

public class ECSScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.


    This .NET example performs the following tasks:
        1. List ECS Cluster ARNs.
```

```
        2. List services in every cluster
        3. List Task ARNs in every cluster.
    */

    private static ILogger logger = null!;
    private static ECSWrapper _ecsWrapper = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .Build();

        ILoggerFactory loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ECSScenario>();

        var loggerECSWarpper = LoggerFactory.Create(builder =>
{ builder.AddConsole(); })
          .CreateLogger<ECSWrapper>();

        var amazonECSClient = new AmazonECSClient();

        _ecsWrapper = new ECSWrapper(amazonECSClient, loggerECSWarpper);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon ECS example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            await ListClusterARNs();
            await ListServiceARNs();
            await ListTaskARNs();
```

```
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// List ECS Cluster ARNs
    /// </summary>
    private static async Task ListClusterARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List Cluster ARNs from ECS.");
        var arns = await _ecsWrapper.GetClusterARNSAsync();

        foreach (var arn in arns)
        {
            Console.WriteLine($"Cluster arn: {arn}");
            Console.WriteLine($"Cluster name: {arn.Split("/").Last()}");
        }

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// List services in every cluster
    /// </summary>
    private static async Task ListServiceARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List Service ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNSAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting services for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));


            var serviceARNs = await _ecsWrapper.GetServiceARNSAsync(clusterARN);
```

```
            foreach (var serviceARN in serviceARNs)
            {
                Console.WriteLine($"Service arn: {serviceARN}");
                Console.WriteLine($"Service name: {serviceARN.Split("/").Last()}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// List tasks in every cluster
    /// </summary>
    private static async Task ListTaskARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. List Task ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNSAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting tasks for cluster name:
 {clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var taskARNs = await _ecsWrapper.GetTaskARNsAsync(clusterARN);

            foreach (var taskARN in taskARNs)
            {
                Console.WriteLine($"Task arn: {taskARN}");
            }
        }
        Console.WriteLine(new string('-', 80));
    }
}
```

Métodos envoltorios a los que el escenario llama para gestionar ECS las acciones de Amazon.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.Logging;
```

```
namespace ECSActions;

public class ECSWrapper
{
    private readonly AmazonECSClient _ecsClient;
    private readonly ILogger<ECSWrapper> _logger;

    /// <summary>
    /// Constructor for the ECS wrapper.
    /// </summary>
    /// <param name="ecsClient">The injected ECS client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public ECSWrapper(AmazonECSClient ecsClient, ILogger<ECSWrapper> logger)

    {
        _logger = logger;
        _ecsClient = ecsClient;
    }

    /// <summary>
    /// List cluster ARNs available.
    /// </summary>
    /// <returns>The ARN list of clusters.</returns>
    public async Task<List<string>> GetClusterARNSAsync()
    {

        Console.WriteLine("Getting a list of all the clusters in your AWS
 account...");
        List<string> clusterArnList = new List<string>();
        // Get a list of all the clusters in your AWS account
        try
        {

            var listClustersResponse = _ecsClient.Paginators.ListClusters(new
 ListClustersRequest
            {
            });

            var clusterArns = listClustersResponse.ClusterArns;

            // Print the ARNs of the clusters
            await foreach (var clusterArn in clusterArns)
            {
```

```
                clusterArnList.Add(clusterArn);
            }

            if (clusterArnList.Count == 0)
            {
                _logger.LogWarning("No clusters found in your AWS account.");
            }
            return clusterArnList;
        }
        catch (Exception e)
        {
            _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
            throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        }
    }

    /// <summary>
    /// List service ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of services in given cluster.</returns>
    public async Task<List<string>> GetServiceARNSAsync(string clusterARN)
    {
        List<string> serviceArns = new List<string>();

        var request = new ListServicesRequest
        {
            Cluster = clusterARN
        };
        // Call the ListServices API operation and get the list of service ARNs
        var serviceList = _ecsClient.Paginators.ListServices(request);

        await foreach (var serviceARN in serviceList.ServiceArns)
        {
            if (serviceARN is null)
                continue;

            serviceArns.Add(serviceARN);
        }

        if (serviceArns.Count == 0)
        {
```

```
            _logger.LogWarning($"No services found in cluster {clusterARN} .");
        }

        return serviceArns;
    }

    /// <summary>
    /// List task ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of tasks in given cluster.</returns>
    public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
    {
        // Set up the request to describe the tasks in the service
        var listTasksRequest = new ListTasksRequest
        {
            Cluster = clusterARN
        };
        List<string> taskArns = new List<string>();

        // Call the ListTasks API operation and get the list of task ARNs
        var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

        await foreach (var task in tasks.TaskArns)
        {
            if (task is null)
                continue;


            taskArns.Add(task);
        }

        if (taskArns.Count == 0)
        {
            _logger.LogWarning("No tasks found in cluster: " + clusterARN);
        }

        return taskArns;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - ListClusters
  - ListServices
  - ListTasks

Ejemplos de Elastic Load Balancing: versión 2 utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET con Elastic Load Balancing, versión 2.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Escenarios

Acciones

`CreateListener`

En el siguiente ejemplo de código, se muestra cómo usar `CreateListener`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });
```

```
                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }
```

- Para API obtener más información, consulte CreateListenerla AWS SDK for .NET APIReferencia.

`CreateLoadBalancer`

En el siguiente ejemplo de código, se muestra cómo usar `CreateLoadBalancer`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });
```

```
                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }
```

- Para API obtener más información, consulte CreateLoadBalancerla AWS SDK for .NET APIReferencia.

`CreateTargetGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateTargetGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
```

```
        return targetGroup;
    }
```

- Para API obtener más información, consulte CreateTargetGroupla AWS SDK for .NET APIReferencia.

`DeleteLoadBalancer`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteLoadBalancer`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
```

```
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }
```

- Para API obtener más información, consulte DeleteLoadBalancerla AWS SDK for .NET APIReferencia.

`DeleteTargetGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTargetGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
```

```
                    });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

- Para API obtener más información, consulte DeleteTargetGroupla AWS SDK for .NET APIReferencia.

`DescribeLoadBalancers`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeLoadBalancers`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
```

```
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }
```

- Para API obtener más información, consulte DescribeLoadBalancersla AWS SDK for .NET APIReferencia.

`DescribeTargetHealth`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeTargetHealth`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }
```

- Para API obtener más información, consulte DescribeTargetHealthla AWS SDK for .NET APIReferencia.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (AmazonEC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias en un rango específico.
- Gestione y distribuya HTTP las solicitudes con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecuta un servidor web Python en cada EC2 instancia para gestionar HTTP las solicitudes. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

AWS SDK for .NET

> **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 el escenario interactivo en un símbolo del sistema.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
```

```
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
```

```
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
```

```
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();
```

```
        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
 The target group\n"
            + "defines how the load balancer connects to instances. The load
 balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
 instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
 _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
 _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
 protocol, port, defaultVpc.VpcId);

        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);
```

```
            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
 ipString);
                }
            }
            loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
 browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
```

```
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
```

```
                            $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                            $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                            "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
 the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
 customer with a somewhat better experience than failure.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
 "static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
 checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
 _smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
 target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
```

```
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");
```

```
        Console.WriteLine($"\nNow, checking target health indicates that the
 instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
 the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
 endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
 rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
 auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
```

```
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
```

```
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
 incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Crea una clase que agrupe las EC2 acciones de Auto Scaling y Amazon.

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
```

```
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
```

```
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
```

```
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
```

```
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
```

```
   /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
   /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       await CreateKeyPair(_keyPairName);
       await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

       var startServerText = await File.ReadAllTextAsync(startupScriptPath);
       var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

       var amiLatest = await _amazonSsm.GetParameterAsync(
           new GetParameterRequest() { Name = _amiParam });
       var amiId = amiLatest.Parameter.Value;
       var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
           new CreateLaunchTemplateRequest()
           {
               LaunchTemplateName = _launchTemplateName,
               LaunchTemplateData = new RequestLaunchTemplateData()
               {
                   InstanceType = _instanceType,
                   ImageId = amiId,
                   IamInstanceProfile =
                       new
                           LaunchTemplateIamInstanceProfileSpecificationRequest()
                       {
                           Name = _instanceProfileName
                       },
                   KeyName = _keyPairName,
                   UserData = System.Convert.ToBase64String(plainTextBytes)
               }
           });
       return launchTemplateResponse.LaunchTemplate;

   }

   /// <summary>
   /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
   /// </summary>
   /// <returns>A list of availability zones.</returns>
```

```
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
 {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
 exists.");
        }
    }
```

```
    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }
```

```
        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
```

```
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
```

```
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
```

```
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
```

```
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }
```

```
            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
```

```
   public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
   {
       var portIsOpen = false;
       foreach (var ipPermission in group.IpPermissions)
       {
           if (ipPermission.FromPort == port)
           {
               foreach (var ipRange in ipPermission.Ipv4Ranges)
               {
                   var cidr = ipRange.CidrIp;
                   if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                   {
                       portIsOpen = true;
                   }
               }

               if (ipPermission.PrefixListIds.Any())
               {
                   portIsOpen = true;
               }

               if (!portIsOpen)
               {
                   Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
               }
               else
               {
                   break;
               }
           }
       }

       return portIsOpen;
   }

   /// <summary>
   /// Add an ingress rule to the specified security group that allows access on
the
   /// specified port from the specified IP address.
   /// </summary>
```

```
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
```

```
    }
}
```

Cree una clase que resuma las acciones de Elastic Load Balancing.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
```

```
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
```

```
                    Names = new List<string>() { groupName }
                });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
 groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
```

```
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
```

```
                    });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
```

```
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
 describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
```

```
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
```

```
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

Cree una clase que utilice DynamoDB para simular un servicio de recomendaciones.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
```

```
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
                            KeyType = KeyType.RANGE
                        }
                    },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");
```

```
            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
```

```
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Cree una clase que agrupe las acciones de Systems Manager.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
```

```
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
```

```
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

EventBridge ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with EventBridge.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola EventBridge

Los siguientes ejemplos de código muestran cómo empezar a usarlo EventBridge.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.EventBridge;
using Amazon.EventBridge.Model;

namespace EventBridgeActions;

public static class HelloEventBridge
{
    static async Task Main(string[] args)
    {
```

```
        var eventBridgeClient = new AmazonEventBridgeClient();

        Console.WriteLine($"Hello Amazon EventBridge! Following are some of your
 EventBuses:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five event buses.
        var response = await eventBridgeClient.ListEventBusesAsync(
            new ListEventBusesRequest()
            {
                Limit = 5
            });

        foreach (var eventBus in response.EventBuses)
        {
            Console.WriteLine($"\tEventBus: {eventBus.Name}");
            Console.WriteLine($"\tArn: {eventBus.Arn}");
            Console.WriteLine($"\tPolicy: {eventBus.Policy}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListEventBusesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crear una regla y agregarle un destino
- Habilitar y deshabilitar reglas.

- Enumerar y actualizar reglas y destinos.
- Enviar eventos y, después, limpiar los recursos

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 EventBridgeScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks with Amazon EventBridge:
    - Create a rule.
    - Add a target to a rule.
    - Enable and disable rules.
    - List rules and targets.
    - Update rules and targets.
    - Send events.
    - Delete the rule.
    */

    private static ILogger logger = null!;
    private static EventBridgeWrapper _eventBridgeWrapper = null!;
    private static IConfiguration _configuration = null!;

    private static IAmazonIdentityManagementService? _iamClient = null!;
    private static IAmazonSimpleNotificationService? _snsClient = null!;
    private static IAmazonS3 _s3Client = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
```

```
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonEventBridge>()
            .AddAWSService<IAmazonIdentityManagementService>()
            .AddAWSService<IAmazonS3>()
            .AddAWSService<IAmazonSimpleNotificationService>()
            .AddTransient<EventBridgeWrapper>()
            )
            .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<EventBridgeScenario>();

        ServicesSetup(host);

        string topicArn = "";
        string roleArn = "";

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon EventBridge example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            roleArn = await CreateRole();

            await CreateBucketWithEventBridgeEvents();

            await AddEventRule(roleArn);

            await ListEventRules();

            topicArn = await CreateSnsTopic();
```

```
            var email = await SubscribeToSnsTopic(topicArn);

            await AddSnsTarget(topicArn);

            await ListTargets();

            await ListRulesForTarget(topicArn);

            await UploadS3File(_s3Client);

            await ChangeRuleState(false);

            await GetRuleState();

            await UpdateSnsEventRule(topicArn);

            await ChangeRuleState(true);

            await UploadS3File(_s3Client);

            await UpdateToCustomRule(topicArn);

            await TriggerCustomRule(email);

            await CleanupResources(topicArn);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources(topicArn);
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The Amazon EventBridge example scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
```

```
        _eventBridgeWrapper =
host.Services.GetRequiredService<EventBridgeWrapper>();
        _snsClient =
host.Services.GetRequiredService<IAmazonSimpleNotificationService>();
        _s3Client = host.Services.GetRequiredService<IAmazonS3>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    }

    /// <summary>
    /// Create a role to be used by EventBridge.
    /// </summary>
    /// <returns>The role Amazon Resource Name (ARN).</returns>
    public static async Task<string> CreateRole()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating a role to use with EventBridge and attaching
managed policy AmazonEventBridgeFullAccess.");
        Console.WriteLine(new string('-', 80));

        var roleName = _configuration["roleName"];

        var assumeRolePolicy = "{" +
                               "\"Version\": \"2012-10-17\"," +
                               "\"Statement\": [{" +
                               "\"Effect\": \"Allow\"," +
                               "\"Principal\": {" +
                               $"\"Service\": \"events.amazonaws.com\"" +
                               "}," +
                               "\"Action\": \"sts:AssumeRole\"" +
                               "}]" +
                               "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = roleName
            });

        await _iamClient.AttachRolePolicyAsync(
            new AttachRolePolicyRequest()
            {
```

```
                PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
                RoleName = roleName
            });
        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        return roleResult.Role.Arn;
    }

    /// <summary>
    /// Create an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge
events enabled.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateBucketWithEventBridgeEvents()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating an S3 bucket with EventBridge events enabled.");

        var testBucketName = _configuration["testBucketName"];

        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            testBucketName);

        if (!bucketExists)
        {
            await _s3Client.PutBucketAsync(new PutBucketRequest()
            {
                BucketName = testBucketName,
                UseClientRegion = true
            });
        }

        await _s3Client.PutBucketNotificationAsync(new
PutBucketNotificationRequest()
        {
            BucketName = testBucketName,
            EventBridgeConfiguration = new EventBridgeConfiguration()
        });

        Console.WriteLine($"\tAdded bucket {testBucketName} with EventBridge events
enabled.");

        Console.WriteLine(new string('-', 80));
```

```
    }

    /// <summary>
    /// Create and upload a file to an S3 bucket to trigger an event.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task UploadS3File(IAmazonS3 s3Client)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Uploading a file to the test bucket. This will trigger a
subscription email.");

        var testBucketName = _configuration["testBucketName"];

        var fileName = $"example_upload_{DateTime.UtcNow.Ticks}.txt";

        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for testing uploads.");
        }

        await s3Client.PutObjectAsync(new PutObjectRequest()
        {
            FilePath = fileName,
            BucketName = testBucketName
        });

        Console.WriteLine($"\tPress Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create an Amazon Simple Notification Service (Amazon SNS) topic to use as an
EventBridge target.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> CreateSnsTopic()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine(
            "Creating an Amazon Simple Notification Service (Amazon SNS) topic for
email subscriptions.");

        var topicName = _configuration["topicName"];

        string topicPolicy = "{" +
                             "\"Version\": \"2012-10-17\"," +
                             "\"Statement\": [{" +
                             "\"Sid\": \"EventBridgePublishTopic\"," +
                             "\"Effect\": \"Allow\"," +
                             "\"Principal\": {" +
                             $"\"Service\": \"events.amazonaws.com\"" +
                             "}," +
                             "\"Resource\": \"*\"," +
                             "\"Action\": \"sns:Publish\"" +
                             "}]" +
                             "}";

        var topicAttributes = new Dictionary<string, string>()
        {
            { "Policy", topicPolicy }
        };

        var topicResponse = await _snsClient!.CreateTopicAsync(new
CreateTopicRequest()
        {
            Name = topicName,
            Attributes = topicAttributes

        });

        Console.WriteLine($"\tAdded topic {topicName} for email subscriptions.");

        Console.WriteLine(new string('-', 80));

        return topicResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a user email to an SNS topic.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>The user's email.</returns>
```

```
    private static async Task<string> SubscribeToSnsTopic(string topicArn)
    {
        Console.WriteLine(new string('-', 80));


        string email = "";
        while (string.IsNullOrEmpty(email))
        {
            Console.WriteLine("Enter your email to subscribe to the Amazon SNS
topic:");
            email = Console.ReadLine()!;
        }

        var subscriptions = new List<string>();
        var paginatedSubscriptions =
_snsClient!.Paginators.ListSubscriptionsByTopic(
            new ListSubscriptionsByTopicRequest()
            {
                TopicArn = topicArn
            });

        // Get the entire list using the paginator.
        await foreach (var subscription in paginatedSubscriptions.Subscriptions)
        {
            subscriptions.Add(subscription.Endpoint);
        }

        if (subscriptions.Contains(email))
        {
            Console.WriteLine($"\tYour email is already subscribed.");
            Console.WriteLine(new string('-', 80));
            return email;
        }

        await _snsClient.SubscribeAsync(new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "email",
            Endpoint = email
        });

        Console.WriteLine($"Use the link in the email you received to confirm your
subscription, then press Enter to continue.");
```

```
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
        return email;
    }

    /// <summary>
    /// Add a rule which triggers when a file is uploaded to an S3 bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role used by EventBridge.</param>
    /// <returns>Async task.</returns>
    private static async Task AddEventRule(string roleArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating an EventBridge event that sends an email when an
Amazon S3 object is created.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];

        await _eventBridgeWrapper.PutS3UploadRule(roleArn, eventRuleName,
testBucketName);
        Console.WriteLine($"\tAdded event rule {eventRuleName} for bucket
{testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an SNS target to the rule.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
    private static async Task AddSnsTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Adding a target to the rule to that sends an email when
the rule is triggered.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];
        var topicName = _configuration["topicName"];
        await _eventBridgeWrapper.AddSnsTargetToRule(eventRuleName, topicArn);
```

```
        Console.WriteLine($"\tAdded event rule {eventRuleName} with Amazon SNS
target {topicName} for bucket {testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the event rules on the default event bus.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListEventRules()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Current event rules:");

        var rules = await _eventBridgeWrapper.ListAllRulesForEventBus();
        rules.ForEach(r => Console.WriteLine($"\tRule: {r.Name} Description:
{r.Description} State: {r.State}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Update the event target to use a transform.
    /// </summary>
    /// <param name="topicArn">The SNS topic ARN target to update.</param>
    /// <returns>Async task.</returns>
    private static async Task UpdateSnsEventRule(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Let's update the event target with a transform.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];

        await
_eventBridgeWrapper.UpdateS3UploadRuleTargetWithTransform(eventRuleName, topicArn);
        Console.WriteLine($"\tUpdated event rule {eventRuleName} with Amazon SNS
target {topicArn} for bucket {testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
```

```
    /// Update the rule to use a custom event pattern.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task UpdateToCustomRule(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Updating the event pattern to be triggered by a custom
event instead.");

        var eventRuleName = _configuration["eventRuleName"];

        await _eventBridgeWrapper.UpdateCustomEventPattern(eventRuleName);

        Console.WriteLine($"\tUpdated event rule {eventRuleName} to custom
pattern.");
        await _eventBridgeWrapper.UpdateCustomRuleTargetWithTransform(eventRuleName,
            topicArn);

        Console.WriteLine($"\tUpdated event target {topicArn}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Send rule events for a custom rule using the user's email address.
    /// </summary>
    /// <param name="email">The email address to include.</param>
    /// <returns>Async task.</returns>
    private static async Task TriggerCustomRule(string email)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Sending an event to trigger the rule. This will trigger a
subscription email.");

        await _eventBridgeWrapper.PutCustomEmailEvent(email);

        Console.WriteLine($"\tEvents have been sent. Press Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the targets for a rule.
```

```
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListTargets()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the targets for a particular rule.");

        var eventRuleName = _configuration["eventRuleName"];
        var targets = await _eventBridgeWrapper.ListAllTargetsOnRule(eventRuleName);
        targets.ForEach(t => Console.WriteLine($"\tTarget: {t.Arn} Id: {t.Id} Input:
{t.Input}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the rules for a particular target.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
    private static async Task ListRulesForTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the rules for a particular target.");

        var rules = await _eventBridgeWrapper.ListAllRuleNamesByTarget(topicArn);
        rules.ForEach(r => Console.WriteLine($"\tRule: {r}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Enable or disable a particular rule.
    /// </summary>
    /// <param name="isEnabled">True to enable the rule, otherwise false.</param>
    /// <returns>Async task.</returns>
    private static async Task ChangeRuleState(bool isEnabled)
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];

        if (!isEnabled)
        {
            Console.WriteLine($"Disabling the rule: {eventRuleName}");
```

```
            await _eventBridgeWrapper.DisableRuleByName(eventRuleName);
        }
        else
        {
            Console.WriteLine($"Enabling the rule: {eventRuleName}");
            await _eventBridgeWrapper.EnableRuleByName(eventRuleName);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the current state of the rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetRuleState()
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];

        var state = await _eventBridgeWrapper.GetRuleStateByRuleName(eventRuleName);
        Console.WriteLine($"Rule {eventRuleName} is in current state {state}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic to clean up.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        var eventRuleName = _configuration["eventRuleName"];
        if (GetYesNoResponse($"\tDelete all targets and event rule {eventRuleName}?
(y/n)"))
        {
            Console.WriteLine($"\tRemoving all targets from the event rule.");
            await _eventBridgeWrapper.RemoveAllTargetsFromRule(eventRuleName);

            Console.WriteLine($"\tDeleting event rule.");
```

```
            await _eventBridgeWrapper.DeleteRuleByName(eventRuleName);
        }

        var topicName = _configuration["topicName"];
        if (GetYesNoResponse($"\tDelete Amazon SNS subscription topic {topicName}?
(y/n)"))
        {
            Console.WriteLine($"\tDeleting topic.");
            await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        }

        var bucketName = _configuration["testBucketName"];
        if (GetYesNoResponse($"\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting bucket.");
            // Delete all objects in the bucket.
            var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.S3Objects
                    .Select(o => new KeyVersion { Key = o.Key }).ToList()
            });
            // Now delete the bucket.
            await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
            {
                BucketName = bucketName
            });
        }

        var roleName = _configuration["roleName"];
        if (GetYesNoResponse($"\tDelete role {roleName}? (y/n)"))
        {
            Console.WriteLine($"\tDetaching policy and deleting role.");

            await _iamClient!.DetachRolePolicyAsync(new DetachRolePolicyRequest()
            {
```

```
                RoleName = roleName,
                PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
            });

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = roleName
            });
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Crea una clase que abarque EventBridge las operaciones.

```
/// <summary>
/// Wrapper for Amazon EventBridge operations.
/// </summary>
public class EventBridgeWrapper
{
    private readonly IAmazonEventBridge _amazonEventBridge;
    private readonly ILogger<EventBridgeWrapper> _logger;

    /// <summary>
    /// Constructor for the EventBridge wrapper.
```

```
    /// </summary>
    /// <param name="amazonEventBridge">The injected EventBridge client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public EventBridgeWrapper(IAmazonEventBridge amazonEventBridge,
ILogger<EventBridgeWrapper> logger)

    {
        _amazonEventBridge = amazonEventBridge;
        _logger = logger;
    }

    /// <summary>
    /// Get the state for a rule by the rule name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
    /// <returns>The state of the rule.</returns>
    public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
    {
        var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
            new DescribeRuleRequest()
            {
                Name = ruleName,
                EventBusName = eventBusName
            });
        return ruleResponse.State;
    }

    /// <summary>
    /// Enable a particular rule on an event bus.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
            new EnableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Disable a particular rule on an event bus.
    /// </summary
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
            new DisableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the rules on an event bus.
    /// </summary>
    /// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
    /// <returns>The list of rules.</returns>
    public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
    {
        var results = new List<Rule>();
        var request = new ListRulesRequest()
        {
            EventBusName = eventBusArn
        };
        // Get all of the pages of rules.
        ListRulesResponse response;
        do
        {
            response = await _amazonEventBridge.ListRulesAsync(request);
            results.AddRange(response.Rules);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
```

```
    /// List all of the targets matching a rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>The list of targets.</returns>
    public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
    {
        var results = new List<Target>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse response;
        do
        {
            response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
            results.AddRange(response.Targets);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// List names of all rules matching a target.
    /// </summary>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <returns>The list of rule names.</returns>
    public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
    {
        var results = new List<string>();
        var request = new ListRuleNamesByTargetRequest()
        {
            TargetArn = targetArn
        };
        ListRuleNamesByTargetResponse response;
        do
        {
            response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
            results.AddRange(response.RuleNames);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);
```

```
        return results;
    }

    /// <summary>
    /// Create a new event rule that triggers when an Amazon S3 object is created in
 a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
 string bucketName)
    {
        string eventPattern = "{" +
                              "\"source\": [\"aws.s3\"]," +
                                  "\"detail-type\": [\"Object Created\"]," +
                                  "\"detail\": {" +
                                      "\"bucket\": {" +
                                          "\"name\": [\"" + bucketName + "\"]" +
                                      "}" +
                                  "}" +
                              "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
 event bus.</param>
```

```
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
                InputTransformer = new InputTransformer()
                {
                    InputPathsMap = new Dictionary<string, string>()
                    {
                        {"bucket", "$.detail.bucket.name"},
                        {"time", "$.time"}
                    },
                    InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\""
                }
            }
        };
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });
        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
        return targetID;
    }

    /// <summary>
```

```
    /// Update a custom rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateCustomRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
                InputTransformer = new InputTransformer()
                {
                    InputTemplate = "\"Notification: sample event was received.\""
                }
            }
        };
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });
        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
        return targetID;
    }

    /// <summary>
```

```
    /// Add an event to the event bus that includes an email, message, and time.
    /// </summary>
    /// <param name="email">The email to use in the event detail of the custom
event.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutCustomEmailEvent(string email)
    {
        var eventDetail = new
        {
            UserEmail = email,
            Message = "This event was generated by example code.",
            UtcTime = DateTime.UtcNow.ToString("g")
        };
        var response = await _amazonEventBridge.PutEventsAsync(
            new PutEventsRequest()
            {
                Entries = new List<PutEventsRequestEntry>()
                {
                    new PutEventsRequestEntry()
                    {
                        Source = "ExampleSource",
                        Detail = JsonSerializer.Serialize(eventDetail),
                        DetailType = "ExampleType"
                    }
                }
            });

        return response.FailedEntryCount == 0;
    }

    /// <summary>
    /// Update a rule to use a custom defined event pattern.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <returns>The ARN of the updated rule.</returns>
    public async Task<string> UpdateCustomEventPattern(string ruleName)
    {
        string customEventsPattern = "{" +
                                     "\"source\": [\"ExampleSource\"]," +
                                     "\"detail-type\": [\"ExampleType\"]" +
                                     "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
```

```
            {
                Name = ruleName,
                Description = "Custom test rule",
                EventPattern = customEventsPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Add an Amazon SNS target topic to a rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <param name="targetArn">The ARN of the Amazon SNS target.</param>
    /// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        // Create the list of targets and add a new target.
        var targets = new List<Target>
        {
            new Target()
            {
                Arn = targetArn,
                Id = targetID
            }
        };

        // Add the targets to the rule.
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });

        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
```

```
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return targetID;
    }

    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
    {
        var targetIds = new List<string>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse targetsResponse;
        do
        {
            targetsResponse = await
_amazonEventBridge.ListTargetsByRuleAsync(request);
            targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
            request.NextToken = targetsResponse.NextToken;

        } while (targetsResponse.NextToken is not null);

        var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
            new RemoveTargetsRequest()
            {
                Rule = ruleName,
                Ids = targetIds
            });

        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
```

```
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
 {e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteRuleByName(string ruleName)
    {
        var response = await _amazonEventBridge.DeleteRuleAsync(
            new DeleteRuleRequest()
            {
                Name = ruleName
            });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - DeleteRule
  - DescribeRule
  - DisableRule
  - EnableRule
  - ListRuleNamesByTarget
  - ListRules
  - ListTargetsByRule
  - PutEvents
  - PutRule
  - PutTargets

Acciones

`DeleteRule`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar una regla por su nombre.

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte DeleteRulela AWS SDK for .NET APIReferencia.

`DescribeRule`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtener el estado de una regla mediante la descripción de la regla.

```
   /// <summary>
   /// Get the state for a rule by the rule name.
   /// </summary>
   /// <param name="ruleName">The name of the rule.</param>
   /// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
   /// <returns>The state of the rule.</returns>
   public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
   {
       var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
           new DescribeRuleRequest()
           {
               Name = ruleName,
               EventBusName = eventBusName
           });
       return ruleResponse.State;
   }
```

- Para API obtener más información, consulte DescribeRulela AWS SDK for .NET APIReferencia.

`DisableRule`

En el siguiente ejemplo de código, se muestra cómo usar `DisableRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Deshabilitar una regla por su nombre de regla.

```
/// <summary>
/// Disable a particular rule on an event bus.
/// </summary
/// <param name="ruleName">The name of the rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableRuleByName(string ruleName)
{
    var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
        new DisableRuleRequest()
        {
            Name = ruleName
        });
    return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte DisableRulela AWS SDK for .NET APIReferencia.

`EnableRule`

En el siguiente ejemplo de código, se muestra cómo usar `EnableRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Habilitar una regla por su nombre de regla.

```
    /// <summary>
    /// Enable a particular rule on an event bus.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
            new EnableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte EnableRulela AWS SDK for .NET APIReferencia.

`ListRuleNamesByTarget`

En el siguiente ejemplo de código, se muestra cómo usar `ListRuleNamesByTarget`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enumerar todos los nombres de regla por el destino.

```
    /// <summary>
    /// List names of all rules matching a target.
    /// </summary>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <returns>The list of rule names.</returns>
    public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
    {
        var results = new List<string>();
        var request = new ListRuleNamesByTargetRequest()
```

```
        {
            TargetArn = targetArn
        };
        ListRuleNamesByTargetResponse response;
        do
        {
            response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
            results.AddRange(response.RuleNames);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }
```

- Para API obtener más información, consulte ListRuleNamesByTargetla AWS SDK for .NET APIReferencia.

`ListRules`

En el siguiente ejemplo de código, se muestra cómo usar `ListRules`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enumerar todas las reglas de un bus de eventos.

```
    /// <summary>
    /// List the rules on an event bus.
    /// </summary>
    /// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
    /// <returns>The list of rules.</returns>
    public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
```

```
    {
        var results = new List<Rule>();
        var request = new ListRulesRequest()
        {
            EventBusName = eventBusArn
        };
        // Get all of the pages of rules.
        ListRulesResponse response;
        do
        {
            response = await _amazonEventBridge.ListRulesAsync(request);
            results.AddRange(response.Rules);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }
```

- Para API obtener más información, consulte ListRulesla AWS SDK for .NET APIReferencia.

`ListTargetsByRule`

En el siguiente ejemplo de código, se muestra cómo usar `ListTargetsByRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enumerar todos los destinos para una regla por el nombre de regla.

```
    /// <summary>
    /// List all of the targets matching a rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>The list of targets.</returns>
    public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
```

```
    {
        var results = new List<Target>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse response;
        do
        {
            response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
            results.AddRange(response.Targets);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }
```

- Para API obtener más información, consulte ListTargetsByRulela AWS SDK for .NET APIReferencia.

`PutEvents`

En el siguiente ejemplo de código, se muestra cómo usar `PutEvents`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enviar un evento que coincida con el patrón personalizado de una regla.

```
    /// <summary>
    /// Add an event to the event bus that includes an email, message, and time.
    /// </summary>
    /// <param name="email">The email to use in the event detail of the custom
event.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> PutCustomEmailEvent(string email)
    {
        var eventDetail = new
        {
            UserEmail = email,
            Message = "This event was generated by example code.",
            UtcTime = DateTime.UtcNow.ToString("g")
        };
        var response = await _amazonEventBridge.PutEventsAsync(
            new PutEventsRequest()
            {
                Entries = new List<PutEventsRequestEntry>()
                {
                    new PutEventsRequestEntry()
                    {
                        Source = "ExampleSource",
                        Detail = JsonSerializer.Serialize(eventDetail),
                        DetailType = "ExampleType"
                    }
                }
            });

        return response.FailedEntryCount == 0;
    }
```

- Para API obtener más información, consulte PutEventsla AWS SDK for .NET APIReferencia.

`PutRule`

En el siguiente ejemplo de código, se muestra cómo usar `PutRule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una regla que se active cuando se agregue un objeto a un bucket de Amazon Simple Storage Service.

```
    /// <summary>
    /// Create a new event rule that triggers when an Amazon S3 object is created in
 a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
 string bucketName)
    {
        string eventPattern = "{" +
                                "\"source\": [\"aws.s3\"]," +
                                    "\"detail-type\": [\"Object Created\"]," +
                                    "\"detail\": {" +
                                        "\"bucket\": {" +
                                            "\"name\": [\"" + bucketName + "\"]" +
                                        "}" +
                                    "}" +
                                "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }
```

Crear una regla que use un patrón personalizado.

```
    /// <summary>
    /// Update a rule to use a custom defined event pattern.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <returns>The ARN of the updated rule.</returns>
    public async Task<string> UpdateCustomEventPattern(string ruleName)
```

```
    {
        string customEventsPattern = "{" +
                                     "\"source\": [\"ExampleSource\"]," +
                                     "\"detail-type\": [\"ExampleType\"]" +
                                     "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Custom test rule",
                EventPattern = customEventsPattern
            });

        return response.RuleArn;
    }
```

- Para API obtener más información, consulte PutRulela AWS SDK for .NET APIReferencia.

`PutTargets`

En el siguiente ejemplo de código, se muestra cómo usar `PutTargets`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Añade un SNS tema de Amazon como objetivo de una regla.

```
    /// <summary>
    /// Add an Amazon SNS target topic to a rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <param name="targetArn">The ARN of the Amazon SNS target.</param>
    /// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
    /// <returns>The ID of the target.</returns>
```

```
    public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        // Create the list of targets and add a new target.
        var targets = new List<Target>
        {
            new Target()
            {
                Arn = targetArn,
                Id = targetID
            }
        };

        // Add the targets to the rule.
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });

        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return targetID;
    }
```

Agregue un transformador de entrada al destino de una regla.

```
    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
```

```
   /// <param name="ruleName">The name of the rule.</param>
   /// <param name="targetArn">The ARN of the target.</param>
   /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
   /// <returns>The ID of the target.</returns>
   public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
   {
       var targetID = Guid.NewGuid().ToString();

       var targets = new List<Target>
       {
           new Target()
           {
               Id = targetID,
               Arn = targetArn,
               InputTransformer = new InputTransformer()
               {
                   InputPathsMap = new Dictionary<string, string>()
                   {
                       {"bucket", "$.detail.bucket.name"},
                       {"time", "$.time"}
                   },
                   InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\""
               }
           }
       };
       var response = await _amazonEventBridge.PutTargetsAsync(
           new PutTargetsRequest()
           {
               EventBusName = eventBusArn,
               Rule = ruleName,
               Targets = targets,
           });
       if (response.FailedEntryCount > 0)
       {
           response.FailedEntries.ForEach(e =>
           {
               _logger.LogError(
                   $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
           });
       }
```

```
        return targetID;
    }
```

- Para API obtener más información, consulta PutTargetsla AWS SDK for .NET APIReferencia.

`RemoveTargets`

En el siguiente ejemplo de código, se muestra cómo usar `RemoveTargets`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar todos los destinos de una regla por el nombre de regla.

```
    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
    {
        var targetIds = new List<string>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse targetsResponse;
        do
        {
            targetsResponse = await
_amazonEventBridge.ListTargetsByRuleAsync(request);
            targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
            request.NextToken = targetsResponse.NextToken;

        } while (targetsResponse.NextToken is not null);
```

```
        var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
            new RemoveTargetsRequest()
            {
                Rule = ruleName,
                Ids = targetIds
            });

        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte RemoveTargetsla AWS SDK for .NET APIReferencia.

EventBridge Ejemplos de planificadores que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for .NET with EventBridge Scheduler.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola, EventBridge Scheduler

El siguiente ejemplo de código muestra cómo empezar a utilizar EventBridge Scheduler.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
public static class HelloScheduler
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the EventBridge Scheduler service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonScheduler>()
            ).Build();

        // Now the client is available for injection.
        var schedulerClient = host.Services.GetRequiredService<IAmazonScheduler>();

        // You can use await and any of the async methods to get a response, or a
 paginator to list schedules or groups.
        var results = new List<ScheduleSummary>();
        var paginateSchedules = schedulerClient.Paginators.ListSchedules(
            new ListSchedulesRequest());
        Console.WriteLine(
            $"Hello AWS Scheduler! Let's list schedules in your account.");
        // Get the entire list using the paginator.
        await foreach (var schedule in paginateSchedules.Schedules)
        {
            results.Add(schedule);
        }
        Console.WriteLine($"\tTotal of {results.Count} schedule(s) available.");
        results.ForEach(s => Console.WriteLine($"\tSchedule: {s.Name}"));
    }
}
```

- Para API obtener más información, consulte ListSchedulesla AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios

Acciones

`CreateSchedule`

En el siguiente ejemplo de código, se muestra cómo usar `CreateSchedule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates a new schedule in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule.</param>
    /// <param name="scheduleExpression">The schedule expression that defines when
 the schedule should run.</param>
    /// <param name="scheduleGroupName">The name of the schedule group to which the
 schedule should be added.</param>
    /// <param name="deleteAfterCompletion">Indicates whether to delete the schedule
 after completion.</param>
    /// <param name="useFlexibleTimeWindow">Indicates whether to use a flexible time
 window for the schedule.</param>
    /// <param name="targetArn">ARN of the event target.</param>
    /// <param name="roleArn">Execution Role ARN.</param>
    /// <returns>True if the schedule was created successfully, false otherwise.</
returns>
    public async Task<bool> CreateScheduleAsync(
```

```
            string name,
            string scheduleExpression,
            string scheduleGroupName,
            string targetArn,
            string roleArn,
            string input,
            bool deleteAfterCompletion = false,
            bool useFlexibleTimeWindow = false)
    {
        try
        {
            int hoursToRun = 1;
            int flexibleTimeWindowMinutes = 10;

            var request = new CreateScheduleRequest
            {
                Name = name,
                ScheduleExpression = scheduleExpression,
                GroupName = scheduleGroupName,
                Target = new Target { Arn = targetArn, RoleArn = roleArn, Input =
input },
                ActionAfterCompletion = deleteAfterCompletion
                    ? ActionAfterCompletion.DELETE
                    : ActionAfterCompletion.NONE,
                StartDate = DateTime.UtcNow, // Ignored for one-time schedules.
                EndDate =
                    DateTime.UtcNow
                        .AddHours(hoursToRun) // Ignored for one-time schedules.
            };
            // Allow a flexible time window if the caller specifies it.
            request.FlexibleTimeWindow = new FlexibleTimeWindow
            {
                Mode = useFlexibleTimeWindow
                    ? FlexibleTimeWindowMode.FLEXIBLE
                    : FlexibleTimeWindowMode.OFF,
                MaximumWindowInMinutes = useFlexibleTimeWindow
                    ? flexibleTimeWindowMinutes
                    : null
            };

            var response = await _amazonScheduler.CreateScheduleAsync(request);

            Console.WriteLine($"Successfully created schedule '{name}' " +
```

```
                                $"in schedule group '{scheduleGroupName}':
{response.ScheduleArn}.");
            return true;
        }
        catch (ConflictException ex)
        {
            // If the name is not unique, a ConflictException will be thrown.
            _logger.LogError($"Failed to create schedule '{name}' due to a conflict.
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating schedule '{name}' "
+
                               $"in schedule group '{scheduleGroupName}':
{ex.Message}");
            return false;
        }
    }
```

- Para API obtener más información, consulte CreateSchedulela AWS SDK for .NET APIReferencia.

`CreateScheduleGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateScheduleGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates a new schedule group in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule group.</param>
```

```
   /// <returns>True if the schedule group was created successfully, false
otherwise.</returns>
   public async Task<bool> CreateScheduleGroupAsync(string name)
   {
       try
       {
           var request = new CreateScheduleGroupRequest { Name = name };

           var response = await _amazonScheduler.CreateScheduleGroupAsync(request);

           Console.WriteLine($"Successfully created schedule group '{name}':
{response.ScheduleGroupArn}.");
           return true;

       }
       catch (ConflictException ex)
       {
           // If the name is not unique, a ConflictException will be thrown.
           _logger.LogError($"Failed to create schedule group '{name}' due to a
conflict. {ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError(
               $"An error occurred while creating schedule group '{name}':
{ex.Message}");
           return false;
       }
   }
```

- Para API obtener más información, consulte CreateScheduleGroupla AWS SDK for .NET APIReferencia.

`DeleteSchedule`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteSchedule`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes an existing schedule from Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule to delete.</param>
    /// <param name="groupName">The group name of the schedule to delete.</param>
    /// <returns>True if the schedule was deleted successfully, false otherwise.</
returns>
    public async Task<bool> DeleteScheduleAsync(string name, string groupName)
    {
        try
        {
            var request = new DeleteScheduleRequest
            {
                Name = name,
                GroupName = groupName
            };

            await _amazonScheduler.DeleteScheduleAsync(request);

            Console.WriteLine($"Successfully deleted schedule with name '{name}'.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError(
                $"Failed to delete schedule with ID '{name}' because the resource
 was not found: {ex.Message}");
            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(
```

```
                $"An error occurred while deleting schedule with ID '{name}':
{ex.Message}");
            return false;
        }
    }
```

- Para API obtener más información, consulte DeleteSchedulela AWS SDK for .NET APIReferencia.

`DeleteScheduleGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteScheduleGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes an existing schedule group from Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule group to delete.</param>
    /// <returns>True if the schedule group was deleted successfully, false
otherwise.</returns>
    public async Task<bool> DeleteScheduleGroupAsync(string name)
    {
        try
        {
            var request = new DeleteScheduleGroupRequest { Name = name };

            await _amazonScheduler.DeleteScheduleGroupAsync(request);

            Console.WriteLine($"Successfully deleted schedule group '{name}'.");
            return true;

        }
        catch (ResourceNotFoundException ex)
```

```
        {
            _logger.LogError(
                $"Failed to delete schedule group '{name}' because the resource was
not found: {ex.Message}");
            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while deleting schedule group '{name}':
{ex.Message}");
            return false;
        }
    }
```

- Para API obtener más información, consulte DeleteScheduleGroupla AWS SDK for .NET APIReferencia.

Escenarios

Flujo de trabajo de eventos programados

En el siguiente ejemplo de código, se muestra cómo:

- Implemente una AWS CloudFormation pila con los recursos necesarios.
- Cree un grupo de EventBridge horarios de Scheduler.
- Cree un cronograma de EventBridge programación único con una ventana de tiempo flexible.
- Cree un horario de EventBridge Scheduler recurrente con una tarifa específica.
- Elimine EventBridge Scheduler, el horario y el grupo de horarios.
- Limpie los recursos y elimine la pila.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecute el flujo de trabajo.

```
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Scheduler;
using Amazon.Scheduler.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using SchedulerActions;
using Exception = System.Exception;

namespace SchedulerScenario;

public class SchedulerWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the Amazon EventBridge
 Scheduler workflow:

    1. Prepare the Application:
       - Prompt the user for an email address to use for the subscription for the
 SNS topic subscription.
       - Prompt the user for a name for the Cloud Formation stack.
       - Deploy the Cloud Formation template in resources/cfn_template.yaml for
 resource creation.
       - Store the outputs of the stack into variables for use in the workflow.
       - Create a schedule group for all workflow schedules.

    2. Create one-time Schedule:
       - Create a one-time schedule to send an initial event.
       - Use a Flexible Time Window and set the schedule to delete after completion.
       - Wait for the user to receive the event email from SNS.

    3. Create a time-based schedule:
       - Prompt the user for how many X times per Y hours a recurring event should
 be scheduled.
       - Create the scheduled event for X times per hour for Y hours.
       - Wait for the user to receive the event email from SNS.
```

```
       - Delete the schedule when the user is finished.

    4. Clean up:
       - Prompt the user for y/n answer if they want to destroy the stack and clean
 up all resources.
       - Delete the schedule group.
       - Destroy the Cloud Formation stack and wait until the stack has been
 removed.
    */

    public static ILogger<SchedulerWorkflow> _logger = null!;
    public static SchedulerWrapper _schedulerWrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _roleArn = null!;
    private static string _snsTopicArn = null!;

    public static bool _interactive = true;
    private static string _stackName = "default-scheduler-workflow-stack-name";
    private static string _scheduleGroupName = "workflow-schedules-group";
    private static string _stackResourcePath = "../../../../../../workflows/
eventbridge_scheduler/resources/cfn_template.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonScheduler>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<SchedulerWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<SchedulerWorkflow>();
```

```
            _schedulerWrapper =
host.Services.GetRequiredService<SchedulerWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon EventBridge Scheduler Workflow.");
        Console.WriteLine(new string('-', 80));

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await CreateOneTimeSchedule();
                Console.WriteLine(new string('-', 80));

                Console.WriteLine(new string('-', 80));
                await CreateRecurringSchedule();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the workflow, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("Amazon EventBridge Scheduler workflow completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
```

```
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        try
        {
            // Prompt the user for an email address to use for the subscription.
            Console.WriteLine("\nThis example creates resources in a CloudFormation
stack, including an SNS topic" +
                              "\nthat will be subscribed to the EventBridge Scheduler
events. " +
                              "\n\nYou will need to confirm the subscription in order to
receive event emails. ");

            var emailAddress = PromptUserForEmail();

            // Prompt the user for a name for the CloudFormation stack
            _stackName = PromptUserForStackName();

            // Deploy the CloudFormation stack
            var deploySuccess = await DeployCloudFormationStack(_stackName,
emailAddress);

            if (deploySuccess)
            {
                // Create a schedule group for all workflow schedules
                await
_schedulerWrapper.CreateScheduleGroupAsync(_scheduleGroupName);

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }
        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
```

```
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="email">The email to use for the subscription.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
string email)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = { Capability.CAPABILITY_NAMED_IAM }
            };

            // If an email is provided, set the parameter.
            if (!string.IsNullOrWhiteSpace(email))
            {
                request.Parameters = new List<Parameter>()
                {
                    new() { ParameterKey = "email", ParameterValue = email }
                };
            }

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                // Wait for the stack to be in CREATE_COMPLETE state
                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    // Retrieve the output values
                    var success = await GetStackOutputs(response.StackId);
                    return success;
                }
```

```
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName, email);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="client">The CloudFormation client.</param>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 10;
        const int retryDelay = 30000; // 30 seconds.

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
```

```
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    Console.WriteLine("CloudFormation stack creation complete.");
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                        describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    Console.WriteLine("CloudFormation stack creation failed.");
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Retrieves the output values from the CloudFormation stack.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    private static async Task<bool> GetStackOutputs(string stackId)
    {
        try
        {
```

```
            var describeStacksRequest = new DescribeStacksRequest { StackName =
stackId };

            var describeStacksResponse =
                await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                var stack = describeStacksResponse.Stacks[0];
                _roleArn = GetStackOutputValue(stack, "RoleARN");
                _snsTopicArn = GetStackOutputValue(stack, "SNStopicARN");
                return true;
            }
            else
            {
                _logger.LogError($"No stack found for stack outputs: {stackId}");
                return false;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(
                ex, $"Failed to retrieve CloudFormation stack outputs: {stackId}");
            return false;
        }
    }

    /// <summary>
    /// Get an output value by key from a CloudFormation stack.
    /// </summary>
    /// <param name="stack">The CloudFormation stack.</param>
    /// <param name="outputKey">The key of the output.</param>
    /// <returns>The value as a string.</returns>
    private static string GetStackOutputValue(Stack stack, string outputKey)
    {
        var output = stack.Outputs.First(o => o.OutputKey == outputKey);
        var outputValue = output.OutputValue;
        Console.WriteLine($"Stack output {outputKey}: {outputValue}");
        return outputValue;
    }

    /// <summary>
    /// Creates a one-time schedule to send an initial event.
```

```
    /// </summary>
    /// <returns>True if the one-time schedule was created successfully.</returns>
    public static async Task<bool> CreateOneTimeSchedule()
    {
        var scheduleName =
            PromptUserForResourceName("Enter a name for the one-time schedule:");

        Console.WriteLine($"Creating a one-time schedule named '{scheduleName}' " +
                          $"\nto send an initial event in 1 minute with a flexible
time window...");
        try
        {
            // Create a one-time schedule with a flexible time
            // window set to delete after completion.
            // You may also set a timezone instead of using UTC.
            var scheduledTime = DateTime.UtcNow.AddMinutes(1).ToString("s");

            var createSuccess = await _schedulerWrapper.CreateScheduleAsync(
                scheduleName,
                $"at({scheduledTime})",
                _scheduleGroupName,
                _snsTopicArn,
                _roleArn,
                $"One time scheduled event test from schedule {scheduleName}.",
                true,
                useFlexibleTimeWindow: true);

            Console.WriteLine($"Subscription email will receive an email from this
event.");
            Console.WriteLine($"You must confirm your subscription to receive event
emails.");

            Console.WriteLine($"One-time schedule '{scheduleName}' created
successfully.");
            return createSuccess;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError(ex, $"The target with ARN '{_snsTopicArn}' was not
found.");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, $"An error occurred while creating the one-time
schedule '{scheduleName}'.");
            return false;
        }
    }

    /// <summary>
    /// Create a recurring schedule to send events at a specified rate in minutes.
    /// </summary>
    /// <returns>True if the recurring schedule was created successfully.</returns>
    public static async Task<bool> CreateRecurringSchedule()
    {
        Console.WriteLine("Creating a recurring schedule to send events for one
hour...");

        try
        {
            // Prompt the user for a schedule name.
            var scheduleName =
                PromptUserForResourceName("Enter a name for the recurring schedule:
");

            // Prompt the user for the schedule rate (in minutes).
            var scheduleRateInMinutes =
                PromptUserForInteger("Enter the desired schedule rate (in minutes):
");

            // Create the recurring schedule.
            var createSuccess = await _schedulerWrapper.CreateScheduleAsync(
                scheduleName,
                $"rate({scheduleRateInMinutes} minutes)",
                _scheduleGroupName,
                _snsTopicArn,
                _roleArn,
                $"Recurrent event test from schedule {scheduleName}.");

            Console.WriteLine($"Subscription email will receive an email from this
event.");
            Console.WriteLine($"You must confirm your subscription to receive event
emails.");

            // Delete the schedule when the user is finished.
            if (!_interactive || GetYesNoResponse($"Are you ready to delete the
'{scheduleName}' schedule? (y/n)"))
```

```
            {
                await _schedulerWrapper.DeleteScheduleAsync(scheduleName,
_scheduleGroupName);
            }

            return createSuccess;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError(ex, $"The target with ARN '{_snsTopicArn}' was not
found.");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while creating the recurring
schedule.");
            return false;
        }
    }

    /// <summary>
    /// Cleans up the resources created during the workflow.
    /// </summary>
    /// <returns>True if the cleanup was successful.</returns>
    public static async Task<bool> Cleanup()
    {
        // Prompt the user to confirm cleanup.
        var cleanup = !_interactive || GetYesNoResponse(
            "Do you want to delete all resources created by this workflow? (y/n) ");
        if (cleanup)
        {
            try
            {
                // Delete the schedule group.
                var groupDeleteSuccess = await
_schedulerWrapper.DeleteScheduleGroupAsync(_scheduleGroupName);

                // Destroy the CloudFormation stack and wait for it to be removed.
                var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);

                return groupDeleteSuccess && stackDeleteSuccess;
            }
```

```
            catch (Exception ex)
            {
                _logger.LogError(ex,
                    "An error occurred while cleaning up the resources.");
                return false;
            }
        }
        _logger.LogInformation("EventBridge Scheduler workflow is complete.");
        return true;
    }

    /// <summary>
    /// Delete the resources in the stack and wait for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the stack.</param>
    /// <param name="forceDelete">True to force delete the stack.</param>
    /// <returns>True if successful.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
    {
        var request = new DeleteStackRequest
        {
            StackName = stackName,
        };

        if (forceDelete)
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{_stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(_stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{_stackName}' has been
deleted.");
            return true;
        }
        else
        {
```

```
            _logger.LogError($"Failed to delete CloudFormation stack
'{_stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Wait for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the stack.</param>
    /// <param name="forceDelete">True to force delete the stack.</param>
    /// <returns>True if successful.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 10;
        const int retryDelay = 30000; // 30 seconds

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
describeStacksResponse.Stacks[0].StackStatus == StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    // Try one time to force delete.
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
```

```
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                // Stack does not exist, so it has been successfully deleted.
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompt the user for a valid email address.
    /// </summary>
    /// <returns>The valid email address.</returns>
    private static string PromptUserForEmail()
    {
        if (_interactive)
        {
            Console.WriteLine("Enter an email address to use for event
subscriptions: ");

            string email = Console.ReadLine()!;
```

```
            if (!IsValidEmail(email))
            {
                Console.WriteLine("Invalid email address. Please try again.");
                return PromptUserForEmail();
            }
            return email;
        }
        // Used when running without user prompts.
        return "";
    }

    /// <summary>
    /// Prompt the user for a non-empty stack name.
    /// </summary>
    /// <returns>The valid stack name</returns>
    private static string PromptUserForStackName()
    {
        Console.WriteLine("Enter a name for the AWS Cloud Formation Stack: ");
        if (_interactive)
        {
            string stackName = Console.ReadLine()!;
            var regex = "[a-zA-Z][-a-zA-Z0-9]|arn:[-a-zA-Z0-9:/._+]";
            if (!Regex.IsMatch(stackName, regex))
            {
                Console.WriteLine(
                    $"Invalid stack name. Please use a name that matches the pattern
{regex}.");
                return PromptUserForStackName();
            }

            return stackName;
        }
        // Used when running without user prompts.
        return _stackName;
    }

    /// <summary>
    /// Prompt the user for a non-empty resource name.
    /// </summary>
    /// <returns>The valid stack name</returns>
    private static string PromptUserForResourceName(string prompt)
    {
        if (_interactive)
```

```
        {
            Console.WriteLine(prompt);
            string resourceName = Console.ReadLine()!;
            var regex = "[0-9a-zA-Z-_.]+";
            if (!Regex.IsMatch(resourceName, regex))
            {
                Console.WriteLine($"Invalid resource name. Please use a name that
matches the pattern {regex}.");
                return PromptUserForResourceName(prompt);
            }
            return resourceName!;
        }
        // Used when running without user prompts.
        return "resource-" + Guid.NewGuid();
    }

    /// <summary>
    /// Prompt the user for a non-empty resource name.
    /// </summary>
    /// <returns>The valid stack name</returns>
    private static int PromptUserForInteger(string prompt)
    {
        if (_interactive)
        {
            Console.WriteLine(prompt);
            string stringResponse = Console.ReadLine()!;
            if (string.IsNullOrWhiteSpace(stringResponse) ||
                !Int32.TryParse(stringResponse, out var intResponse))
            {
                Console.WriteLine($"Invalid integer. ");
                return PromptUserForInteger(prompt);
            }
            return intResponse!;
        }
        // Used when running without user prompts.
        return 1;
    }

    /// <summary>
    /// Use System Mail to check for a valid email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email address.</returns>
    private static bool IsValidEmail(string email)
```

```
    {
        try
        {
            var mailAddress = new System.Net.Mail.MailAddress(email);
            return mailAddress.Address == email;
        }
        catch
        {
            // Invalid emails will cause an exception, return false.
            return false;
        }
    }
}
```

Envoltorio para operaciones de servicio.

```
using Amazon.Scheduler;
using Amazon.Scheduler.Model;
using Microsoft.Extensions.Logging;

namespace SchedulerActions;

/// <summary>
/// Wrapper class for Amazon EventBridge Scheduler operations.
/// </summary>
public class SchedulerWrapper
{
    private readonly IAmazonScheduler _amazonScheduler;
    private readonly ILogger<SchedulerWrapper> _logger;

    /// <summary>
    /// Constructor for the SchedulerWrapper class.
    /// </summary>
    /// <param name="amazonScheduler">The injected EventBridge Scheduler client.</
param>
    /// <param name="logger">The injected logger.</param>
    public SchedulerWrapper(IAmazonScheduler amazonScheduler,
 ILogger<SchedulerWrapper> logger)
    {
        _amazonScheduler = amazonScheduler;
        _logger = logger;
    }
```

```
    /// <summary>
    /// Creates a new schedule in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule.</param>
    /// <param name="scheduleExpression">The schedule expression that defines when
 the schedule should run.</param>
    /// <param name="scheduleGroupName">The name of the schedule group to which the
 schedule should be added.</param>
    /// <param name="deleteAfterCompletion">Indicates whether to delete the schedule
 after completion.</param>
    /// <param name="useFlexibleTimeWindow">Indicates whether to use a flexible time
 window for the schedule.</param>
    /// <param name="targetArn">ARN of the event target.</param>
    /// <param name="roleArn">Execution Role ARN.</param>
    /// <returns>True if the schedule was created successfully, false otherwise.</
returns>
    public async Task<bool> CreateScheduleAsync(
            string name,
            string scheduleExpression,
            string scheduleGroupName,
            string targetArn,
            string roleArn,
            string input,
            bool deleteAfterCompletion = false,
            bool useFlexibleTimeWindow = false)
    {
        try
        {
            int hoursToRun = 1;
            int flexibleTimeWindowMinutes = 10;

            var request = new CreateScheduleRequest
            {
                Name = name,
                ScheduleExpression = scheduleExpression,
                GroupName = scheduleGroupName,
                Target = new Target { Arn = targetArn, RoleArn = roleArn, Input =
 input },
                ActionAfterCompletion = deleteAfterCompletion
                    ? ActionAfterCompletion.DELETE
                    : ActionAfterCompletion.NONE,
                StartDate = DateTime.UtcNow, // Ignored for one-time schedules.
                EndDate =
```

```
                    DateTime.UtcNow
                        .AddHours(hoursToRun) // Ignored for one-time schedules.
            };
            // Allow a flexible time window if the caller specifies it.
            request.FlexibleTimeWindow = new FlexibleTimeWindow
            {
                Mode = useFlexibleTimeWindow
                    ? FlexibleTimeWindowMode.FLEXIBLE
                    : FlexibleTimeWindowMode.OFF,
                MaximumWindowInMinutes = useFlexibleTimeWindow
                    ? flexibleTimeWindowMinutes
                    : null
            };

            var response = await _amazonScheduler.CreateScheduleAsync(request);

            Console.WriteLine($"Successfully created schedule '{name}' " +
                              $"in schedule group '{scheduleGroupName}':
{response.ScheduleArn}.");
            return true;
        }
        catch (ConflictException ex)
        {
            // If the name is not unique, a ConflictException will be thrown.
            _logger.LogError($"Failed to create schedule '{name}' due to a conflict.
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while creating schedule '{name}' "
+
                             $"in schedule group '{scheduleGroupName}':
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Creates a new schedule group in Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule group.</param>
    /// <returns>True if the schedule group was created successfully, false
otherwise.</returns>
```

```
    public async Task<bool> CreateScheduleGroupAsync(string name)
    {
        try
        {
            var request = new CreateScheduleGroupRequest { Name = name };

            var response = await _amazonScheduler.CreateScheduleGroupAsync(request);

            Console.WriteLine($"Successfully created schedule group '{name}':
 {response.ScheduleGroupArn}.");
            return true;

        }
        catch (ConflictException ex)
        {
            // If the name is not unique, a ConflictException will be thrown.
            _logger.LogError($"Failed to create schedule group '{name}' due to a
 conflict. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while creating schedule group '{name}':
 {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an existing schedule from Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule to delete.</param>
    /// <param name="groupName">The group name of the schedule to delete.</param>
    /// <returns>True if the schedule was deleted successfully, false otherwise.</
returns>
    public async Task<bool> DeleteScheduleAsync(string name, string groupName)
    {
        try
        {
            var request = new DeleteScheduleRequest
            {
                Name = name,
                GroupName = groupName
```

```
            };

            await _amazonScheduler.DeleteScheduleAsync(request);

            Console.WriteLine($"Successfully deleted schedule with name '{name}'.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError(
                $"Failed to delete schedule with ID '{name}' because the resource
was not found: {ex.Message}");
            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while deleting schedule with ID '{name}':
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an existing schedule group from Amazon EventBridge Scheduler.
    /// </summary>
    /// <param name="name">The name of the schedule group to delete.</param>
    /// <returns>True if the schedule group was deleted successfully, false
otherwise.</returns>
    public async Task<bool> DeleteScheduleGroupAsync(string name)
    {
        try
        {
            var request = new DeleteScheduleGroupRequest { Name = name };

            await _amazonScheduler.DeleteScheduleGroupAsync(request);

            Console.WriteLine($"Successfully deleted schedule group '{name}'.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
```

```
            _logger.LogError(
                $"Failed to delete schedule group '{name}' because the resource was
 not found: {ex.Message}");
            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(
                $"An error occurred while deleting schedule group '{name}':
 {ex.Message}");
            return false;
        }
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateSchedule
  - CreateScheduleGroup
  - DeleteSchedule
  - DeleteScheduleGroups

AWS Glue ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with AWS Glue.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola AWS Glue

En los siguientes ejemplos de código se muestra cómo empezar a utilizar AWS Glue.

AWS SDK for .NET

> **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 GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
```

```
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
        while (request.NextToken is not null);

        Console.Clear();
        Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
        if (jobNames.Count == 0)
        {
            Console.WriteLine("You don't have any AWS Glue jobs.");
        }
        else
        {
            jobNames.ForEach(Console.WriteLine);
        }
    }
}
```

- Para API obtener más información, consulte ListJobsla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Cree un rastreador que rastree un bucket público de Amazon S3 y genere una base de datos de CSV metadatos con formato.
- Enumere la información sobre las bases de datos y las tablas en su. AWS Glue Data Catalog
- Cree una tarea para extraer CSV datos del depósito de S3, transformarlos y cargar la salida JSON con formato en otro depósito de S3.

- Incluir información sobre las ejecuciones de trabajos, ver algunos de los datos transformados y limpiar los recursos.

Para obtener más información, consulte el tutorial: Primeros pasos con AWS Glue Studio.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Cree una clase que agrupe AWS Glue las funciones que se utilizan en el escenario.

```
using System.Net;

namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
```

```
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
```

```
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
```

```
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }


    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }
```

```
    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }


    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
 { JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }


    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
```

```
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }


    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }


    /// <summary>
```

```
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }


    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
```

```
        string inputTable,
        string bucketName)
    {
        var request = new StartJobRunRequest
        {
            JobName = jobName,
            Arguments = new Dictionary<string, string>
            {
                {"--input_database", inputDatabase},
                {"--input_table", inputTable},
                {"--output_bucket_url", $"s3://{bucketName}/"}
            }
        };

        var response = await _amazonGlue.StartJobRunAsync(request);
        return response.JobRunId;
    }

}
```

Crear una clase que ejecute el escenario.

```
global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
```

```
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonGlue>()
            .AddTransient<GlueWrapper>()
            .AddTransient<UiWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<GlueBasics>();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // These values are stored in settings.json
        // Once you have run the CDK script to deploy the resources,
        // edit the file to set "BucketName", "RoleName", and "ScriptURL"
        // to the appropriate values. Also set "CrawlerName" to the name
        // you want to give the crawler when it is created.
        string bucketName = _configuration["BucketName"]!;
        string bucketUrl = _configuration["BucketUrl"]!;
        string crawlerName = _configuration["CrawlerName"]!;
        string roleName = _configuration["RoleName"]!;
        string sourceData = _configuration["SourceData"]!;
        string dbName = _configuration["DbName"]!;
        string cron = _configuration["Cron"]!;
        string scriptUrl = _configuration["ScriptURL"]!;
        string jobName = _configuration["JobName"]!;
```

```
        var wrapper = host.Services.GetRequiredService<GlueWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UiWrapper>();

        uiWrapper.DisplayOverview();
        uiWrapper.PressEnter();

        // Create the crawler and wait for it to be ready.
        uiWrapper.DisplayTitle("Create AWS Glue crawler");
        Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

        var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
        var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
        if (crawlerCreated)
        {
            Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
        }
        else
        {
            Console.WriteLine($"Couldn't create crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.DisplayTitle("Start AWS Glue crawler");
        Console.WriteLine("Now let's wait until the crawler has successfully
started.");
        var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
        if (crawlerStarted)
        {
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
```

```
            Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
        }
        else
        {
            Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.PressEnter();

        Console.WriteLine($"\nLet's take a look at the database: {dbName}");
        var database = await wrapper.GetDatabaseAsync(dbName);

        if (database != null)
        {
            uiWrapper.DisplayTitle($"{database.Name} Details");
            Console.WriteLine($"{database.Name} created on {database.CreateTime}");
            Console.WriteLine(database.Description);
        }

        uiWrapper.PressEnter();

        var tables = await wrapper.GetTablesAsync(dbName);
        if (tables.Count > 0)
        {
            tables.ForEach(table =>
            {
                Console.WriteLine($"{table.Name}\tCreated:
{table.CreateTime}\tUpdated: {table.UpdateTime}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create AWS Glue job");
        Console.WriteLine("Creating a new AWS Glue job.");
        var description = "An AWS Glue job created using the AWS SDK for .NET";
        await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Starting AWS Glue job");
        Console.WriteLine("Starting the new AWS Glue job...");
```

```
        var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
        var jobRunComplete = false;
        var jobRun = new JobRun();
        do
        {
            jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
            if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState == "STOPPED"
||
                jobRun.JobRunState == "FAILED" || jobRun.JobRunState == "TIMEOUT")
            {
                jobRunComplete = true;
            }
        } while (!jobRunComplete);

        uiWrapper.DisplayTitle($"Data in {bucketName}");

        // Get the list of data stored in the S3 bucket.
        var s3Client = new AmazonS3Client();

        var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
        response.S3Objects.ForEach(s3Object =>
        {
            Console.WriteLine(s3Object.Key);
        });

        uiWrapper.DisplayTitle("AWS Glue jobs");
        var jobNames = await wrapper.ListJobsAsync();
        jobNames.ForEach(jobName =>
        {
            Console.WriteLine(jobName);
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Get AWS Glue job run information");
        Console.WriteLine("Getting information about the AWS Glue job.");
        var jobRuns = await wrapper.GetJobRunsAsync(jobName);

        jobRuns.ForEach(jobRun =>
        {

Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
```

```
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Deleting resources");
        Console.WriteLine("Deleting the AWS Glue job used by the example.");
        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });

        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);

        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}


namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the URL
 to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
```

```
        Console.WriteLine("\t 3. Get the database created by the crawler and the
tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
```

```
        Console.WriteLine(SepBar);
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Acciones

`CreateCrawler`

En el siguiente ejemplo de código, se muestra cómo usar `CreateCrawler`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
 S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
```

```
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte CreateCrawlerla AWS SDK for .NET APIReferencia.

`CreateJob`

En el siguiente ejemplo de código, se muestra cómo usar `CreateJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
```

```
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte CreateJobla AWS SDK for .NET APIReferencia.

`DeleteCrawler`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteCrawler`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteCrawlerla AWS SDK for .NET APIReferencia.

`DeleteDatabase`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDatabase`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
```

```
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteDatabasela AWS SDK for .NET APIReferencia.

`DeleteJob`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteJobla AWS SDK for .NET APIReferencia.

`DeleteTable`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteTablela AWS SDK for .NET APIReferencia.

`GetCrawler`

En el siguiente ejemplo de código, se muestra cómo usar `GetCrawler`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }
```

- Para API obtener más información, consulte GetCrawlerla AWS SDK for .NET APIReferencia.

`GetDatabase`

En el siguiente ejemplo de código, se muestra cómo usar `GetDatabase`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }
```

- Para API obtener más información, consulte GetDatabasela AWS SDK for .NET APIReferencia.

`GetJobRun`

En el siguiente ejemplo de código, se muestra cómo usar `GetJobRun`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
```

```
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }
```

- Para API obtener más información, consulte GetJobRunla AWS SDK for .NET APIReferencia.

`GetJobRuns`

En el siguiente ejemplo de código, se muestra cómo usar `GetJobRuns`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
```

```
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

- Para API obtener más información, consulte GetJobRunsla AWS SDK for .NET APIReferencia.

`GetTables`

En el siguiente ejemplo de código, se muestra cómo usar `GetTables`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }
```

```
        return tables;
    }
```

- Para API obtener más información, consulte GetTablesla AWS SDK for .NET APIReferencia.

`ListJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }
```

- Para API obtener más información, consulte ListJobsla AWS SDK for .NET APIReferencia.

`StartCrawler`

En el siguiente ejemplo de código, se muestra cómo usar `StartCrawler`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte StartCrawlerla AWS SDK for .NET APIReferencia.

`StartJobRun`

En el siguiente ejemplo de código, se muestra cómo usar `StartJobRun`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- Para API obtener más información, consulte StartJobRunla AWS SDK for .NET APIReferencia.

IAMejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET withIAM.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a IAM

Los siguientes ejemplos de código muestran cómo empezar a usarloIAM.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();

        var listPoliciesPaginator = iamClient.Paginators.ListPolicies(new
 ListPoliciesRequest());
```

```
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
 {policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- Para API obtener más información, consulte ListPolicieslа AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios

Acciones

`AddUserToGroup`

En el siguiente ejemplo de código, se muestra cómo usar `AddUserToGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
```

```
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte AddUserToGroupla AWS SDK for .NET APIReferencia.

`AttachRolePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `AttachRolePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
```

```
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte AttachRolePolicyla AWS SDK for .NET APIReferencia.

`CreateAccessKey`

En el siguiente ejemplo de código, se muestra cómo usar `CreateAccessKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });
```

```
        return response.AccessKey;

    }
```

- Para API obtener más información, consulte CreateAccessKeyla AWS SDK for .NET APIReferencia.

`CreateGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }
```

- Para API obtener más información, consulte CreateGroupla AWS SDK for .NET APIReferencia.

`CreateInstanceProfile`

En el siguiente ejemplo de código, se muestra cómo usar `CreateInstanceProfile`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create a policy, role, and profile that is associated with instances with a
specified name.
   /// An instance's associated profile defines a role that is assumed by the
   /// instance.The role has attached policies that specify the AWS permissions
granted to
   /// clients that run on the instance.
   /// </summary>
   /// <param name="policyName">Name to use for the policy.</param>
   /// <param name="roleName">Name to use for the role.</param>
   /// <param name="profileName">Name to use for the profile.</param>
   /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
   /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
   /// <returns>The Arn of the profile.</returns>
   public async Task<string> CreateInstanceProfileWithName(
       string policyName,
       string roleName,
       string profileName,
       string ssmOnlyPolicyFile,
       List<string>? awsManagedPolicies = null)
   {

       var assumeRoleDoc = "{" +
                                   "\"Version\": \"2012-10-17\"," +
                                   "\"Statement\": [{" +
                                        "\"Effect\": \"Allow\"," +
                                        "\"Principal\": {" +
                                        "\"Service\": [" +
                                            "\"ec2.amazonaws.com\"" +
                                        "]" +
                                        "}," +
                                   "\"Action\": \"sts:AssumeRole\"" +
                                   "}]" +
```

```
                        "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
```

```
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
```

```
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }
```

- Para API obtener más información, consulte CreateInstanceProfilela AWS SDK for .NET APIReferencia.

`CreatePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `CreatePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
```

```
            PolicyName = policyName,
        });

        return response.Policy;
    }
```

- Para API obtener más información, consulte CreatePolicyla AWS SDK for .NET APIReferencia.

`CreateRole`

En el siguiente ejemplo de código, se muestra cómo usar `CreateRole`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }
```

- Para API obtener más información, consulte CreateRolela AWS SDK for .NET APIReferencia.

`CreateServiceLinkedRole`

En el siguiente ejemplo de código, se muestra cómo usar `CreateServiceLinkedRole`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }
```

- Para API obtener más información, consulte CreateServiceLinkedRolela AWS SDK for .NET APIReferencia.

`CreateUser`

En el siguiente ejemplo de código, se muestra cómo usar `CreateUser`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }
```

- Para API obtener más información, consulte CreateUserla AWS SDK for .NET APIReferencia.

`DeleteAccessKey`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteAccessKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteAccessKeyla AWS SDK for .NET APIReferencia.

`DeleteGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteGroupla AWS SDK for .NET APIReferencia.

`DeleteGroupPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteGroupPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
    }
```

- Para API obtener más información, consulte DeleteGroupPolicyla AWS SDK for .NET APIReferencia.

`DeleteInstanceProfile`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteInstanceProfile`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
```

```
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }
```

- Para API obtener más información, consulte DeleteInstanceProfilela AWS SDK for .NET APIReferencia.

`DeletePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DeletePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeletePolicyla AWS SDK for .NET APIReferencia.

`DeleteRole`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteRole`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM role.
```

```
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteRolela AWS SDK for .NET APIReferencia.

`DeleteRolePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteRolePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteRolePolicyla AWS SDK for .NET APIReferencia.

`DeleteUser`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteUser`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteUserla AWS SDK for .NET APIReferencia.

`DeleteUserPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteUserPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteUserPolicyla AWS SDK for .NET APIReferencia.

`DetachRolePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DetachRolePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DetachRolePolicyla AWS SDK for .NET APIReferencia.

`GetAccountPasswordPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `GetAccountPasswordPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
```

```
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }
```

- Para API obtener más información, consulte GetAccountPasswordPolicyla AWS SDK for .NET APIReferencia.

`GetPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `GetPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }
```

- Para API obtener más información, consulte GetPolicyla AWS SDK for .NET APIReferencia.

`GetRole`

En el siguiente ejemplo de código, se muestra cómo usar `GetRole`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}
```

- Para API obtener más información, consulte GetRolela AWS SDK for .NET APIReferencia.

`GetUser`

En el siguiente ejemplo de código, se muestra cómo usar `GetUser`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }
```

- Para API obtener más información, consulte GetUserla AWS SDK for .NET APIReferencia.

`ListAttachedRolePolicies`

En el siguiente ejemplo de código, se muestra cómo usar `ListAttachedRolePolicies`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
```

```
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }
```

- Para API obtener más información, consulte ListAttachedRolePoliciesla AWS SDK for .NET APIReferencia.

`ListGroups`

En el siguiente ejemplo de código, se muestra cómo usar `ListGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
```

```
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }
```

- Para API obtener más información, consulte ListGroupsla AWS SDK for .NET APIReferencia.

`ListPolicies`

En el siguiente ejemplo de código, se muestra cómo usar `ListPolicies`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
```

```
        }

        return policies;
    }
```

- Para API obtener más información, consulte ListPolicieslа AWS SDK for .NET APIReferencia.

`ListRolePolicies`

En el siguiente ejemplo de código, se muestra cómo usar `ListRolePolicies`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }
```

- Para API obtener más información, consulte ListRolePoliciesla AWS SDK for .NET APIReferencia.

`ListRoles`

En el siguiente ejemplo de código, se muestra cómo usar `ListRoles`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }
```

- Para API obtener más información, consulte ListRolesla AWS SDK for .NET APIReferencia.

`ListSAMLProviders`

En el siguiente ejemplo de código, se muestra cómo usar `ListSAMLProviders`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }
```

- Para API obtener más información, consulte L istSAMLProviders en la AWS SDK for .NET APIreferencia.

`ListUsers`

En el siguiente ejemplo de código, se muestra cómo usar `ListUsers`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
```

```
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }
```

- Para API obtener más información, consulte ListUsersla AWS SDK for .NET APIReferencia.

`PutGroupPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `PutGroupPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
    {
        var request = new PutGroupPolicyRequest
```

```
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte PutGroupPolicyla AWS SDK for .NET APIReferencia.

`PutRolePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `PutRolePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
```

```
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte PutRolePolicyla AWS SDK for .NET APIReferencia.

`RemoveUserFromGroup`

En el siguiente ejemplo de código, se muestra cómo usar `RemoveUserFromGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };
```

```
        var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte RemoveUserFromGroupla AWS SDK for .NET APIReferencia.

Escenarios

Creación y administración de un servicio resiliente

El siguiente ejemplo de código muestra cómo crear un servicio web con equilibrio de carga que muestre recomendaciones de libros, películas y canciones. El ejemplo muestra cómo responde el servicio a los errores y cómo reestructurarlo para aumentar la resiliencia cuando se produzcan errores.

- Utilice un grupo de Amazon EC2 Auto Scaling para crear instancias de Amazon Elastic Compute Cloud (AmazonEC2) basadas en una plantilla de lanzamiento y para mantener el número de instancias en un rango específico.
- Gestione y distribuya HTTP las solicitudes con Elastic Load Balancing.
- Supervise el estado de las instancias de un grupo de escalado automático y reenvíe las solicitudes solo a las instancias en buen estado.
- Ejecuta un servidor web Python en cada EC2 instancia para gestionar HTTP las solicitudes. El servidor web responde con recomendaciones y comprobaciones de estado.
- Simule un servicio de recomendaciones con una tabla de Amazon DynamoDB.
- Controle la respuesta del servidor web a las solicitudes y las comprobaciones de estado actualizando AWS Systems Manager los parámetros.

AWS SDK for .NET

> **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 el escenario interactivo en un símbolo del sistema.

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
```

```
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
```

```
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
```

```
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
```

```
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);
```

```
        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
```

```
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
                }
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }
```

```
    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
 different parts of the system\n" +
                          "to create situations where the web service fails, and
 shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
 these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
 recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
 recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();
```

```
        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
```

```
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
```

```
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
```

```
   /// <returns>Async task.</returns>
   public static async Task<bool> DestroyResources(bool interactive)
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine(
           "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
           "that were created for this demo."
       );

       if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
       {
           await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
           await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
           await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
           await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
           await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
           await _autoScalerWrapper.DeleteInstanceProfile(
               _autoScalerWrapper.BadCredsProfileName,
               _autoScalerWrapper.BadCredsRoleName
           );
           await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
       }
       else
       {
           Console.WriteLine(
               "Ok, we'll leave the resources intact.\n" +
               "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
           );
       }

       Console.WriteLine(new string('-', 80));
       return true;
   }
```

Crea una clase que agrupe las EC2 acciones de Auto Scaling y Amazon.

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
```

```
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {
```

```
        var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                     "\"Service\": [" +
                                         "\"ec2.amazonaws.com\"" +
                                     "]" +
                                     "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }
```

```
            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
```

```
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
```

```
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);
```

```
        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
```

```
   public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
   {
       try
       {
           await _amazonAutoScaling.CreateAutoScalingGroupAsync(
               new CreateAutoScalingGroupRequest()
               {
                   AutoScalingGroupName = groupName,
                   AvailabilityZones = availabilityZones,
                   LaunchTemplate =
                       new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                       {
                           LaunchTemplateName = _launchTemplateName,
                           Version = "$Default"
                       },
                   MaxSize = groupSize,
                   MinSize = groupSize
               });
           Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
       }
       catch (EntityAlreadyExistsException)
       {
           Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
       }
   }

   /// <summary>
   /// Get the default VPC for the account.
   /// </summary>
   /// <returns>The default VPC object.</returns>
   public async Task<Vpc> GetDefaultVpc()
   {
       var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
           new DescribeVpcsRequest()
           {
               Filters = new List<Amazon.EC2.Model.Filter>()
               {
                   new ("is-default", new List<string>() { "true" })
               }
           });
       return vpcResponse.Vpcs[0];
```

```
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
```

```
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
```

```
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
```

```
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);
```

```
            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
```

```
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
```

```
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
```

```
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }
```

```
                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                      "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
```

```
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Cree una clase que resuma las acciones de Elastic Load Balancing.

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
```

```
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }
```

```
    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
```

```
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
 subnets
    /// and forwards requests to the specified target group.
    /// </summary>
```

```
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });

               var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

               loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
           }
           catch (LoadBalancerNotFoundException)
           {
               loadBalancerReady = false;
           }
           Thread.Sleep(10000);
       }
       // Create the listener.
       await _amazonElasticLoadBalancingV2.CreateListenerAsync(
           new CreateListenerRequest()
```

```
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
```

```
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
 {groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

Cree una clase que utilice DynamoDB para simular un servicio de recomendaciones.

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
```

```
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
```

```
                        AttributeName = "ItemId",
                        AttributeType = ScalarAttributeType.N
                    }
                },
            KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement()
                    {
                        AttributeName = "MediaType",
                        KeyType = KeyType.HASH
                    },
                    new KeySchemaElement()
                    {
                        AttributeName = "ItemId",
                        KeyType = KeyType.RANGE
                    }
                },
            ProvisionedThroughput = new ProvisionedThroughput()
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5
            }
        };
        await _amazonDynamoDb.CreateTableAsync(createRequest);

        // Wait until the table is ACTIVE and then report success.
        Console.Write("\nWaiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = tableName
        };

        TableStatus status;
        do
        {
            Thread.Sleep(2000);

            var describeTableResponse = await
_amazonDynamoDb.DescribeTableAsync(request);
            status = describeTableResponse.Table.TableStatus;

            Console.Write(".");
        }
```

```
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
```

```
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Cree una clase que agrupe las acciones de Systems Manager.

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
```

```
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 true });
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer

- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Creación de un grupo y adición de un usuario

En el siguiente ejemplo de código, se muestra cómo:

- Cree un grupo y concédale todos los permisos de acceso a Amazon S3.
- Cree un nuevo usuario sin permisos para acceder a Amazon S3.
- Agregue el usuario al grupo, muestre que ahora tiene permisos para Amazon S3 y, a continuación, limpie los recursos.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
```

```
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });
```

```
        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }


    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }


    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
```

```
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
 rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }


    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
```

```
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
```

```
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
   /// <summary>
   /// Delete an IAM role policy.
   /// </summary>
   /// <param name="roleName">The name of the IAM role.</param>
   /// <param name="policyName">The name of the IAM role policy to delete.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
   {
       var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
       {
           PolicyName = policyName,
           RoleName = roleName,
       });

       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }


   /// <summary>
   /// Delete an IAM user.
   /// </summary>
   /// <param name="userName">The username of the IAM user to delete.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteUserAsync(string userName)
   {
       var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }


   /// <summary>
   /// Delete an IAM user policy.
   /// </summary>
   /// <param name="policyName">The name of the IAM policy to delete.</param>
   /// <param name="userName">The username of the IAM user.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
   {
```

```
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
 GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
```

```
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }


    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
```

```
   public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
   {
       var attachedPolicies = new List<AttachedPolicyType>();
       var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

       await foreach (var response in attachedRolePoliciesPaginator.Responses)
       {
           attachedPolicies.AddRange(response.AttachedPolicies);
       }

       return attachedPolicies;
   }


   /// <summary>
   /// List IAM groups.
   /// </summary>
   /// <returns>A list of IAM groups.</returns>
   public async Task<List<Group>> ListGroupsAsync()
   {
       var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
       var groups = new List<Group>();

       await foreach (var response in groupsPaginator.Responses)
       {
           groups.AddRange(response.Groups);
       }

       return groups;
   }


   /// <summary>
   /// List IAM policies.
   /// </summary>
   /// <returns>A list of the IAM policies.</returns>
   public async Task<List<ManagedPolicy>> ListPoliciesAsync()
   {
       var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
```

```
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
```

```
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
 ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }


    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
 _IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
```

```
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
 string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
```

```
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}


using Microsoft.Extensions.Configuration;

namespace IAMGroups;

public class IAMGroups
{
    private static ILogger logger = null!;

    // Represents JSON code for AWS full access policy for Amazon Simple
    // Storage Service (Amazon S3).
    private const string S3FullAccessPolicyDocument = "{" +
        " \"Statement\" : [{" +
            " \"Action\" : [\"s3:*\"]," +
            " \"Effect\" : \"Allow\"," +
            " \"Resource\" : \"*\"" +
        "}]" +
    "}";
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the AWS service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
           services.AddAWSService<IAmazonIdentityManagementService>()
           .AddTransient<IAMWrapper>()
           .AddTransient<UIWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
           .CreateLogger<IAMGroups>();

       IConfiguration configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
           .AddJsonFile("settings.json") // Load test settings from .json file.
           .AddJsonFile("settings.local.json",
               true) // Optionally load local settings.
           .Build();

       var groupUserName = configuration["GroupUserName"];
       var groupName = configuration["GroupName"];
       var groupPolicyName = configuration["GroupPolicyName"];
       var groupBucketName = configuration["GroupBucketName"];

       var wrapper = host.Services.GetRequiredService<IAMWrapper>();
       var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

       uiWrapper.DisplayGroupsOverview();
       uiWrapper.PressEnter();

       // Create an IAM group.
       uiWrapper.DisplayTitle("Create IAM group");
       Console.WriteLine("Let's begin by creating a new IAM group.");
       var group = await wrapper.CreateGroupAsync(groupName);
```

```
        // Add an inline IAM policy to the group.
        uiWrapper.DisplayTitle("Add policy to group");
        Console.WriteLine("Add an inline policy to the group that allows members to
have full access to");
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) buckets.");

        await wrapper.PutGroupPolicyAsync(group.GroupName, groupPolicyName,
S3FullAccessPolicyDocument);

        uiWrapper.PressEnter();

        // Now create a new user.
        uiWrapper.DisplayTitle("Create an IAM user");
        Console.WriteLine("Now let's create a new IAM user.");
        var groupUser = await wrapper.CreateUserAsync(groupUserName);

        // Add the new user to the group.
        uiWrapper.DisplayTitle("Add the user to the group");
        Console.WriteLine("Adding the user to the group, which will give the user
the same permissions as the group.");
        await wrapper.AddUserToGroupAsync(groupUser.UserName, group.GroupName);

        Console.WriteLine($"User, {groupUser.UserName}, has been added to the group,
{group.GroupName}.");
        uiWrapper.PressEnter();

        Console.WriteLine("Now that we have created a user, and added the user to
the group, let's create an IAM access key.");

        // Create access and secret keys for the user.
        var accessKey = await wrapper.CreateAccessKeyAsync(groupUserName);
        Console.WriteLine("Key created.");
        uiWrapper.WaitABit(15, "Waiting for the access key to be ready for use.");

        uiWrapper.DisplayTitle("List buckets");
        Console.WriteLine("To prove that the user has access to Amazon S3, list the
S3 buckets for the account.");

        var s3Client = new AmazonS3Client(accessKey.AccessKeyId,
accessKey.SecretAccessKey);
        var stsClient = new AmazonSecurityTokenServiceClient(accessKey.AccessKeyId,
accessKey.SecretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client, stsClient);
```

```
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
            });
        }

        // Show that the user also has write access to Amazon S3 by creating
        // a new bucket.
        uiWrapper.DisplayTitle("Create a bucket");
        Console.WriteLine("Since group members have full access to Amazon S3, let's
create a bucket.");
        var success = await s3Wrapper.PutBucketAsync(groupBucketName);

        if (success)
        {
            Console.WriteLine($"Successfully created the bucket:
{groupBucketName}.");
        }

        uiWrapper.PressEnter();

        Console.WriteLine("Let's list the user's S3 buckets again to show the new
bucket.");

        buckets = await s3Wrapper.ListMyBucketsAsync();

        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Clean up resources");
```

```
        Console.WriteLine("First delete the bucket we created.");
        await s3Wrapper.DeleteBucketAsync(groupBucketName);

        Console.WriteLine($"Now remove the user, {groupUserName}, from the group,
 {groupName}.");
        await wrapper.RemoveUserFromGroupAsync(groupUserName, groupName);

        Console.WriteLine("Delete the user's access key.");
        await wrapper.DeleteAccessKeyAsync(accessKey.AccessKeyId, groupUserName);

        // Now we can safely delete the user.
        Console.WriteLine("Now we can delete the user.");
        await wrapper.DeleteUserAsync(groupUserName);

        uiWrapper.PressEnter();

        Console.WriteLine("Now we will delete the IAM policy attached to the
 group.");
        await wrapper.DeleteGroupPolicyAsync(groupName, groupPolicyName);

        Console.WriteLine("Now we delete the IAM group.");
        await wrapper.DeleteGroupAsync(groupName);

        uiWrapper.PressEnter();

        Console.WriteLine("The IAM groups demo has completed.");

        uiWrapper.PressEnter();
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;
```

```
    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
```

```
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
```

```
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
 (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
 access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
 creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
```

```
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.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();
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - AddUserToGroup
  - AttachRolePolicy
  - CreateAccessKey
  - CreateGroup
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeleteGroup

- DeleteGroupPolicy
- DeleteUser
- PutGroupPolicy
- RemoveUserFromGroup

Crear un usuario y asumir un rol

En el siguiente ejemplo de código, se muestra cómo crear un usuario y asumir un rol.

> **Warning**
>
> Para evitar riesgos de seguridad, no utilice a IAM los usuarios para autenticarse cuando desarrolle software específico o trabaje con datos reales. En cambio, utilice la federación con un proveedor de identidades como AWS IAM Identity Center.

- Crear un usuario que no tenga permisos.
- Crear un rol que conceda permiso para enumerar los buckets de Amazon S3 para la cuenta.
- Agregar una política para que el usuario asuma el rol.
- Asumir el rol y enumerar los buckets de S3 con credenciales temporales, y después limpiar los recursos.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
```

```
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
 AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }
```

```
    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }


    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }


    /// <summary>
    /// Create an IAM service-linked role.
```

```
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
 { UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
 userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
 DeleteAccessKeyRequest
```

```
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
```

```
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
 { UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
 userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
```

```
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
```

```
            RoleName = roleName,
        });

        return response.Role;
    }


    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }


    /// <summary>
    /// List IAM groups.
```

```
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }


    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
```

```
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }
```

```
    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
 _IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
```

```
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
 string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
```

```
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}



using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<IAMWrapper>()
            .AddTransient<UIWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();


        IConfiguration configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();
```

```
        // Values needed for user, role, and policies.
        string userName = configuration["UserName"]!;
        string s3PolicyName = configuration["S3PolicyName"]!;
        string roleName = configuration["RoleName"]!;


        var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayBasicsOverview();
        uiWrapper.PressEnter();

        // First create a user. By default, the new user has
        // no permissions.
        uiWrapper.DisplayTitle("Create User");
        Console.WriteLine($"Creating a new user with user name: {userName}.");
        var user = await iamWrapper.CreateUserAsync(userName);
        var userArn = user.Arn;

        Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
        uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

        // Define a role policy document that allows the new user
        // to assume the role.
        string assumeRolePolicyDocument = "{" +
          "\"Version\": \"2012-10-17\"," +
          "\"Statement\": [{" +
              "\"Effect\": \"Allow\"," +
              "\"Principal\": {" +
              $" \"AWS\": \"{userArn}\"" +
              "}," +
              "\"Action\": \"sts:AssumeRole\"" +
          "}]" +
        "}";

        // Permissions to list all buckets.
        string policyDocument = "{" +
            "\"Version\": \"2012-10-17\"," +
            " \"Statement\" : [{" +
                " \"Action\" : [\"s3:ListAllMyBuckets\"]," +
                " \"Effect\" : \"Allow\"," +
                " \"Resource\" : \"*\"" +
```

```
            "}]" +
        "}";

        // Create an AccessKey for the user.
        uiWrapper.DisplayTitle("Create access key");
        Console.WriteLine("Now let's create an access key for the new user.");
        var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

        var accessKeyId = accessKey.AccessKeyId;
        var secretAccessKey = accessKey.SecretAccessKey;

        Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

        Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
        var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

        // Now try listing the Amazon Simple Storage Service (Amazon S3)
        // buckets. This should fail at this point because the user doesn't
        // have permissions to perform this task.
        uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
        Console.WriteLine("Now let's try to display a list of the user's Amazon S3
buckets.");
        var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
        var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        Console.WriteLine(buckets is null
            ? "As expected, the call to list the buckets has returned a null list."
            : "Something went wrong. This shouldn't have worked.");

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create IAM role");
        Console.WriteLine($"Creating the role: {roleName}");

        // Creating an IAM role to allow listing the S3 buckets. A role name
        // is not case sensitive and must be unique to the account for which it
        // is created.
```

```
        var roleArn = await iamWrapper.CreateRoleAsync(roleName,
 assumeRolePolicyDocument);

        uiWrapper.PressEnter();

        // Create a policy with permissions to list S3 buckets.
        uiWrapper.DisplayTitle("Create IAM policy");
        Console.WriteLine($"Creating the policy: {s3PolicyName}");
        Console.WriteLine("with permissions to list the Amazon S3 buckets for the
 account.");
        var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
 policyDocument);

        // Wait 15 seconds for the IAM policy to be available.
        uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

        // Attach the policy to the role you created earlier.
        uiWrapper.DisplayTitle("Attach new IAM policy");
        Console.WriteLine("Now let's attach the policy to the role.");
        await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

        // Wait 15 seconds for the role to be updated.
        Console.WriteLine();
        uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

        // Use the AWS Security Token Service (AWS STS) to have the user
        // assume the role we created.
        var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
 secretAccessKey);

        // Wait for the new credentials to become valid.
        uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

        var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

        // Try again to list the buckets using the client created with
        // the new user's credentials. This time, it should work.
        var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

        s3Wrapper.UpdateClients(s3Client2, stsClient2);

        buckets = await s3Wrapper.ListMyBucketsAsync();
```

```
        uiWrapper.DisplayTitle("List Amazon S3 buckets");
        Console.WriteLine("This time we should have buckets to list.");
        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName} created:
 {bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        // Now clean up all the resources used in the example.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Thank you for watching. The IAM Basics demo is
 complete.");
        Console.WriteLine("Please wait while we clean up the resources we
 created.");

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
 patience.");
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
```

```
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
 roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
```

```
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
 (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
 access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
 creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }
```

```
    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.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();
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey

- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Ejemplos de Amazon Keyspaces que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Amazon Keyspaces.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon Keyspaces

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon Keyspaces.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace KeyspacesActions;
```

```
public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache Cassandra).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient = host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- Para API obtener más información, consulte ListKeyspacesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crear un espacio de claves y una tabla. El esquema de la tabla contiene datos de películas y tiene habilitada point-in-time la recuperación.
- Conéctese al espacio de claves mediante una TLS conexión segura con autenticación SigV4.
- Consultar la tabla. Agregar, recuperar y actualizar datos de películas.
- Actualizar la tabla. Añadir una columna para llevar un seguimiento de las películas vistas.
- Restaurar la tabla a su estado anterior y limpiar los recursos.

AWS SDK for .NET

> **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 System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using Newtonsoft.Json;


namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
```

```
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonKeyspaces>()
            .AddTransient<KeyspacesWrapper>()
            .AddTransient<CassandraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var keyspacesWrapper = host.Services.GetRequiredService<KeyspacesWrapper>();
        var uiMethods = new UiMethods();

        var keyspaceName = configuration["KeyspaceName"];
        var tableName = configuration["TableName"];

        bool success; // Used to track the results of some operations.

        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the keyspace.
        var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);
```

```
        // Wait for the keyspace to be available. GetKeyspace results in a
        // resource not found error until it is ready for use.
        try
        {
            var getKeyspaceArn = "";
            Console.Write($"Created {keyspaceName}. Waiting for it to become
available. ");
            do
            {
                getKeyspaceArn = await keyspacesWrapper.GetKeyspace(keyspaceName);
                Console.Write(". ");
            } while (getKeyspaceArn != keyspaceArn);
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine("Waiting for keyspace to be created.");
        }

        Console.WriteLine($"\nThe keyspace {keyspaceName} is ready for use.");

        uiMethods.PressEnter();

        // Create the table.
        // First define the schema.
        var allColumns = new List<ColumnDefinition>
        {
            new ColumnDefinition { Name = "title", Type = "text" },
            new ColumnDefinition { Name = "year", Type = "int" },
            new ColumnDefinition { Name = "release_date", Type = "timestamp" },
            new ColumnDefinition { Name = "plot", Type = "text" },
        };

        var partitionKeys = new List<PartitionKey>
        {
            new PartitionKey { Name = "year", },
            new PartitionKey { Name = "title" },
        };

        var tableSchema = new SchemaDefinition
        {
            AllColumns = allColumns,
            PartitionKeys = partitionKeys,
        };
```

```
        var tableArn = await keyspacesWrapper.CreateTable(keyspaceName, tableSchema,
tableName);

        // Wait for the table to be active.
        try
        {
            var resp = new GetTableResponse();
            Console.Write("Waiting for the new table to be active. ");
            do
            {
                try
                {
                    resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
                    Console.Write(".");
                }
                catch (ResourceNotFoundException)
                {
                    Console.Write(".");
                }
            } while (resp.Status != TableStatus.ACTIVE);

            // Display the table's schema.
            Console.WriteLine($"\nTable {tableName} has been created in
{keyspaceName}");
            Console.WriteLine("Let's take a look at the schema.");
            uiMethods.DisplayTitle("All columns");
            resp.SchemaDefinition.AllColumns.ForEach(column =>
            {
                Console.WriteLine($"{column.Name,-40}\t{column.Type,-20}");
            });

            uiMethods.DisplayTitle("Cluster keys");
            resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
            {

Console.WriteLine($"{clusterKey.Name,-40}\t{clusterKey.OrderBy,-20}");
            });

            uiMethods.DisplayTitle("Partition keys");
            resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
            {
                Console.WriteLine($"{partitionKey.Name}");
            });
```

```
            uiMethods.PressEnter();
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        // Access Apache Cassandra using the Cassandra drive for C#.
        var cassandraWrapper = host.Services.GetRequiredService<CassandraWrapper>();
        var movieFilePath = configuration["MovieFile"];

        Console.WriteLine("Let's add some movies to the table we created.");
        var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

        uiMethods.PressEnter();

        Console.WriteLine("Added the following movies to the table:");
        var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
        uiMethods.DisplayTitle("All Movies");

        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            var plot = row.GetValue<string>("plot");
            var release_date = row.GetValue<DateTime>("release_date");
            Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
            Console.WriteLine(uiMethods.SepBar);
        }

        // Update the table schema
        uiMethods.DisplayTitle("Update table schema");
        Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

        // First save the current time as a UTC Date so the original
        // table can be restored later.
        var timeChanged = DateTime.UtcNow;

        // Now update the schema.
        var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
```

```
        uiMethods.PressEnter();

        Console.WriteLine("Now let's mark some of the movies as watched.");

        // Pick some files to mark as watched.
        var movieToWatch = rows[2].GetValue<string>("title");
        var watchedMovieYear = rows[2].GetValue<int>("year");
        var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[6].GetValue<string>("title");
        watchedMovieYear = rows[6].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[9].GetValue<string>("title");
        watchedMovieYear = rows[9].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[10].GetValue<string>("title");
        watchedMovieYear = rows[10].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[13].GetValue<string>("title");
        watchedMovieYear = rows[13].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        uiMethods.DisplayTitle("Watched movies");
        Console.WriteLine("These movies have been marked as watched:");
        rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            Console.WriteLine($"{title,-40}\t{year,8}");
        }
        uiMethods.PressEnter();

        Console.WriteLine("We can restore the table to its previous state but that
can take up to 20 minutes to complete.");
        string answer;
```

```
        do
        {
            Console.WriteLine("Do you want to restore the table? (y/n)");
            answer = Console.ReadLine();
        } while (answer.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            var restoredTableName = $"{tableName}_restored";
            var restoredTableArn = await keyspacesWrapper.RestoreTable(
                keyspaceName,
                tableName,
                restoredTableName,
                timeChanged);
            // Loop and call GetTable until the table is gone. Once it has been
            // deleted completely, GetTable will raise a ResourceNotFoundException.
            bool wasRestored = false;

            try
            {
                do
                {
                    var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                    wasRestored = (resp.Status == TableStatus.ACTIVE);
                } while (!wasRestored);
            }
            catch (ResourceNotFoundException)
            {
                // If the restored table raised an error, it isn't
                // ready yet.
                Console.Write(".");
            }
        }

        uiMethods.DisplayTitle("Clean up resources.");

        // Delete the table.
        success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

        Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
        Console.WriteLine("Waiting for the table to be removed completely. ");
```

```
        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasDeleted = false;

        try
        {
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
            } while (!wasDeleted);
        }
        catch (ResourceNotFoundException ex)
        {
            wasDeleted = true;
            Console.WriteLine($"{ex.Message} indicates that the table has been
 deleted.");
        }

        // Delete the keyspace.
        success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
        Console.WriteLine("The keyspace has been deleted and the demo is now
 complete.");
    }
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
```

```
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
 schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
 PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }


    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
```

```
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
```

```
   public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
   {
       var response = await _amazonKeyspaces.GetTableAsync(
           new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
       return response;
   }


   /// <summary>
   /// Lists all keyspaces for the account.
   /// </summary>
   /// <returns>Async task.</returns>
   public async Task ListKeyspaces()
   {
       var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

       Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
       Console.WriteLine(new string('-', Console.WindowWidth));
       await foreach (var keyspace in paginator.Keyspaces)
       {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
       }
   }


   /// <summary>
   /// Lists the Amazon Keyspaces tables in a keyspace.
   /// </summary>
   /// <param name="keyspaceName">The name of the keyspace.</param>
   /// <returns>A list of TableSummary objects.</returns>
   public async Task<List<TableSummary>> ListTables(string keyspaceName)
   {
       var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
       response.Tables.ForEach(table =>
       {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
       });
```

```
        return response.Tables;
    }


    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }


    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
```

```
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }

}
```

```
using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
 database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/repository/
sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();
```

```
        _localPathToFile = Path.GetTempPath();

        // Get the Starfield digital certificate and save it locally.
        var client = new WebClient();
        client.DownloadFile(_certLocation, $"{_localPathToFile}/{_certFileName}");

        //var httpClient = new HttpClient();
        //var httpResult = httpClient.Get(fileUrl);
        //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
        //using var fileStream = File.Create(pathToSave);
        //resultStream.CopyTo(fileStream);

        _certCollection = new X509Certificate2Collection();
        _amazoncert = new X509Certificate2($"{_localPathToFile}/{_certFileName}");

        // Get the user name and password stored in the configuration file.
        _userName = _configuration["UserName"]!;
        _pwd = _configuration["Password"]!;

        // For a list of Service Endpoints for Amazon Keyspaces, see:
        // https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
        var awsEndpoint = _configuration["ServiceEndpoint"];

        _cluster = Cluster.Builder()
            .AddContactPoints(awsEndpoint)
            .WithPort(9142)
            .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
            .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
            .WithQueryOptions(
                new QueryOptions()
                    .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
                    .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
            .Build();
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the Apache Cassandra table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A list of movie objects.</returns>
```

```
    public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport =
0)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();

        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        // If numToImport = 0, return all movies in the collection.
        if (numToImport == 0)
        {
            // Now return the entire list of movies.
            return allMovies;
        }
        else
        {
            // Now return the first numToImport entries.
            return allMovies.GetRange(0, numToImport);
        }
    }

    /// <summary>
    /// Insert movies into the movie table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="movieTableName">The Amazon Keyspaces table.</param>
    /// <param name="movieFilePath">The path to the resource file containing
    /// movie data to insert into the table.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
    {
        // Get some movie data from the movies.json file
        var movies = ImportMoviesFromJson(movieFilePath, numToImport);

        var session = _cluster.Connect(keyspaceName);

        string insertCql;
```

```
        RowSet rs;

        // Now we insert the numToImport movies into the table.
        foreach (var movie in movies)
        {
            // Escape single quote characters in the plot.
            insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values($${movie.Title}$$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}', $${movie.Info.Plot}$$)";
            rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
        }

        return true;
    }

    /// <summary>
    /// Gets all of the movies in the movies table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing movie data.</returns>
    public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }

    /// <summary>
    /// Mark a movie in the movie table as watched.
    /// </summary>
```

```
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="title">The title of the movie to mark as watched.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A set of rows containing the changed data.</returns>
    public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
 tableName, string title, int year)
    {
        var session = _cluster.Connect();
        string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
 WHERE title = $${title}$$ AND year = {year};";
        var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
        var rows = rs.GetRows().ToList();
        return rows;
    }

    /// <summary>
    /// Retrieve the movies in the movies table where watched is true.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing information about movies
    /// where watched is true.</returns>
    public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
 tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT title,
 year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW FILTERING"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Acciones

`CreateKeyspace`

En el siguiente ejemplo de código, se muestra cómo usar `CreateKeyspace`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
```

```
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }
```

- Para API obtener más información, consulte CreateKeyspacela AWS SDK for .NET APIReferencia.

`CreateTable`

En el siguiente ejemplo de código, se muestra cómo usar `CreateTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
 schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
 PointInTimeRecoveryStatus.ENABLED }
```

```
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }
```

- Para API obtener más información, consulte CreateTablela AWS SDK for .NET APIReferencia.

`DeleteKeyspace`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteKeyspace`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteKeyspacela AWS SDK for .NET APIReferencia.

`DeleteTable`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteTablela AWS SDK for .NET APIReferencia.

`GetKeyspace`

En el siguiente ejemplo de código, se muestra cómo usar `GetKeyspace`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }
```

- Para API obtener más información, consulte GetKeyspacela AWS SDK for .NET APIReferencia.

`GetTable`

En el siguiente ejemplo de código, se muestra cómo usar `GetTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
```

```
    }
```

- Para API obtener más información, consulte GetTablela AWS SDK for .NET APIReferencia.

`ListKeyspaces`

En el siguiente ejemplo de código, se muestra cómo usar `ListKeyspaces`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Lists all keyspaces for the account.
   /// </summary>
   /// <returns>Async task.</returns>
   public async Task ListKeyspaces()
   {
       var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

       Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
       Console.WriteLine(new string('-', Console.WindowWidth));
       await foreach (var keyspace in paginator.Keyspaces)
       {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
       }
   }
```

- Para API obtener más información, consulte ListKeyspacesla AWS SDK for .NET APIReferencia.

`ListTables`

En el siguiente ejemplo de código, se muestra cómo usar `ListTables`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }
```

- Para API obtener más información, consulte ListTablesla AWS SDK for .NET APIReferencia.

`RestoreTable`

En el siguiente ejemplo de código, se muestra cómo usar `RestoreTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }
```

- Para API obtener más información, consulte RestoreTablela AWS SDK for .NET APIReferencia.

`UpdateTable`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateTable`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
        KeyspaceName = keyspaceName,
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
```

- Para API obtener más información, consulte UpdateTablela AWS SDK for .NET APIReferencia.

Ejemplos de Kinesis que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el AWS SDK for .NET uso de Kinesis.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Ejemplos sin servidor

Acciones

`AddTagsToStream`

En el siguiente ejemplo de código, se muestra cómo usar `AddTagsToStream`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        var tags = new Dictionary<string, string>
        {
```

```
                { "Project", "Sample Kinesis Project" },
                { "Application", "Sample Kinesis App" },
            };

            var success = await ApplyTagsToStreamAsync(client, streamName, tags);

            if (success)
            {
                Console.WriteLine($"Taggs successfully added to {streamName}.");
            }
            else
            {
                Console.WriteLine("Tags were not added to the stream.");
            }
        }

        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Para API obtener más información, consulte AddTagsToStreamla AWS SDK for .NET APIReferencia.

`CreateStream`

En el siguiente ejemplo de código, se muestra cómo usar `CreateStream`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// This example shows how to create a new Amazon Kinesis stream.
    /// </summary>
    public class CreateStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamName = "AmazonKinesisStream";
            int shardCount = 1;

            var success = await CreateNewStreamAsync(client, streamName,
shardCount);
            if (success)
            {
                Console.WriteLine($"The stream, {streamName} successfully
created.");
            }
```

```
        }

        /// <summary>
        /// Creates a new Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client.</param>
        /// <param name="streamName">The name for the new stream.</param>
        /// <param name="shardCount">The number of shards the new stream will
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis client,
 string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Para API obtener más información, consulte CreateStreamla AWS SDK for .NET APIReferencia.

`DeleteStream`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteStream`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to delete an Amazon Kinesis stream.
    /// </summary>
    public class DeleteStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string streamName = "AmazonKinesisStream";

            var success = await DeleteStreamAsync(client, streamName);

            if (success)
            {
                Console.WriteLine($"Stream, {streamName} successfully deleted.");
            }
            else
            {
                Console.WriteLine("Stream not deleted.");
            }
        }

        /// <summary>
        /// Deletes a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the string to delete.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
```

```
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Para API obtener más información, consulte DeleteStreamla AWS SDK for .NET APIReferencia.

`DeregisterStreamConsumer`

En el siguiente ejemplo de código, se muestra cómo usar `DeregisterStreamConsumer`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

```
    /// <summary>
    /// Shows how to deregister a consumer from an Amazon Kinesis stream.
    /// </summary>
    public class DeregisterConsumer
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
```

```
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
                ConsumerARN = consumerARN,
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Para API obtener más información, consulte DeregisterStreamConsumerla AWS SDK for .NET APIReferencia.

`ListStreamConsumers`

En el siguiente ejemplo de código, se muestra cómo usar `ListStreamConsumers`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// List the consumers of an Amazon Kinesis stream.
/// </summary>
public class ListConsumers
{
```

```
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN, maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
            else
            {
                Console.WriteLine("No consumers found.");
            }
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>> ListConsumersAsync(IAmazonKinesis
 client, string streamARN, int maxResults)
        {
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
```

```
    }
```

- Para API obtener más información, consulte ListStreamConsumersla AWS SDK for .NET APIReferencia.

`ListStreams`

En el siguiente ejemplo de código, se muestra cómo usar `ListStreams`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Retrieves and displays a list of existing Amazon Kinesis streams.
/// </summary>
public class ListStreams
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        var response = await client.ListStreamsAsync(new ListStreamsRequest());

        List<string> streamNames = response.StreamNames;

        if (streamNames.Count > 0)
        {
            streamNames
                .ForEach(s => Console.WriteLine($"Stream name: {s}"));
```

```
            }
            else
            {
                Console.WriteLine("No streams were found.");
            }
        }
    }
```

- Para API obtener más información, consulte ListStreamsla AWS SDK for .NET APIReferencia.

`ListTagsForStream`

En el siguiente ejemplo de código, se muestra cómo usar `ListTagsForStream`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to list the tags that have been attached to an Amazon Kinesis
/// stream.
/// </summary>
public class ListTags
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        await ListTagsAsync(client, streamName);
```

```
        }

        /// <summary>
        /// List the tags attached to a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the Kinesis stream for which you
        /// wish to display tags.</param>
        public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
        {
            var request = new ListTagsForStreamRequest
            {
                StreamName = streamName,
                Limit = 10,
            };

            var response = await client.ListTagsForStreamAsync(request);
            DisplayTags(response.Tags);

            while (response.HasMoreTags)
            {
                request.ExclusiveStartTagKey = response.Tags[response.Tags.Count -
1].Key;
                response = await client.ListTagsForStreamAsync(request);
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value: {t.Value}"));
        }
    }
```

- Para API obtener más información, consulte ListTagsForStreamla AWS SDK for .NET APIReferencia.

`RegisterStreamConsumer`

En el siguiente ejemplo de código, se muestra cómo usar `RegisterStreamConsumer`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// This example shows how to register a consumer to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class RegisterConsumer
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
```

```
        /// <param name="consumerName">A string representing the consumer.</param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };

            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- Para API obtener más información, consulte RegisterStreamConsumerla AWS SDK for .NET APIReferencia.

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir registros de un flujo de Kinesis. La función recupera la carga útil de Kinesis, la decodifica desde Base64 y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de Kinesis con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
```

```
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de un flujo de Kinesis. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de errores en los artículos de lote de Kinesis con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
 immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
```

```
                        new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
 record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

AWS KMS ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with AWS KMS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones

Acciones

`CreateAlias`

En el siguiente ejemplo de código, se muestra cómo usar `CreateAlias`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Creates an alias for an AWS Key Management Service (AWS KMS) key.
/// </summary>
public class CreateAlias
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The alias name must start with alias/ and can be
        // up to 256 alphanumeric characters long.
        var aliasName = "alias/ExampleAlias";

        // The value supplied as the TargetKeyId can be either
        // the key ID or key Amazon Resource Name (ARN) of the
```

```
            // AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

            var request = new CreateAliasRequest
            {
                AliasName = aliasName,
                TargetKeyId = keyId,
            };

            var response = await client.CreateAliasAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Alias, {aliasName}, successfully created.");
            }
            else
            {
                Console.WriteLine($"Could not create alias.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateAliasla AWS SDK for .NET APIReferencia.

`CreateGrant`

En el siguiente ejemplo de código, se muestra cómo usar `CreateGrant`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();
```

```
            // The identity that is given permission to perform the operations
            // specified in the grant.
            var grantee = "arn:aws:iam::111122223333:role/ExampleRole";

            // The identifier of the AWS KMS key to which the grant applies. You
            // can use the key ID or the Amazon Resource Name (ARN) of the KMS key.
            var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";

            var request = new CreateGrantRequest
            {
                GranteePrincipal = grantee,
                KeyId = keyId,

                // A list of operations that the grant allows.
                Operations = new List<string>
                {
                    "Encrypt",
                    "Decrypt",
                },
            };

            var response = await client.CreateGrantAsync(request);

            string grantId = response.GrantId; // The unique identifier of the
grant.
            string grantToken = response.GrantToken; // The grant token.

            Console.WriteLine($"Id: {grantId}, Token: {grantToken}");
        }
    }
```

- Para API obtener más información, consulte CreateGrantla AWS SDK for .NET APIReferencia.

`CreateKey`

En el siguiente ejemplo de código, se muestra cómo usar `CreateKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to create a new AWS Key Management Service (AWS KMS)
    /// key.
    /// </summary>
    public class CreateKey
    {
        public static async Task Main()
        {
            // Note that if you need to create a Key in an AWS Region
            // other than the Region defined for the default user, you need to
            // pass the Region to the client constructor.
            var client = new AmazonKeyManagementServiceClient();

            // The call to CreateKeyAsync will create a symmetrical AWS KMS
            // key. For more information about symmetrical and asymmetrical
            // keys, see:
            //
            // https://docs.aws.amazon.com/kms/latest/developerguide/symm-asymm-
choose.html
            var response = await client.CreateKeyAsync(new CreateKeyRequest());

            // The KeyMetadata object contains information about the new AWS KMS
 key.
            KeyMetadata keyMetadata = response.KeyMetadata;

            if (keyMetadata is not null)
            {
                Console.WriteLine($"KMS Key: {keyMetadata.KeyId} was successfully
 created.");
```

```
            }
            else
            {
                Console.WriteLine("Could not create KMS Key.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateKeyla AWS SDK for .NET APIReferencia.

`DescribeKey`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Retrieve information about an AWS Key Management Service (AWS KMS) key.
/// You can supply either the key Id or the key Amazon Resource Name (ARN)
/// to the DescribeKeyRequest KeyId property.
/// </summary>
public class DescribeKey
{
    public static async Task Main()
    {
        var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
        var request = new DescribeKeyRequest
        {
```

```
                KeyId = keyId,
            };

            var client = new AmazonKeyManagementServiceClient();

            var response = await client.DescribeKeyAsync(request);
            var metadata = response.KeyMetadata;

            Console.WriteLine($"{metadata.KeyId} created on:
{metadata.CreationDate}");
            Console.WriteLine($"State: {metadata.KeyState}");
            Console.WriteLine($"{metadata.Description}");
        }
    }
```

- Para API obtener más información, consulte DescribeKeyla AWS SDK for .NET APIReferencia.

`DisableKey`

En el siguiente ejemplo de código, se muestra cómo usar `DisableKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Disable an AWS Key Management Service (AWS KMS) key and then retrieve
    /// the key's status to show that it has been disabled.
    /// </summary>
    public class DisableKey
```

```
    {
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();

            // The identifier of the AWS KMS key to disable. You can use the
            // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

            var request = new DisableKeyRequest
            {
                KeyId = keyId,
            };

            var response = await client.DisableKeyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                // Retrieve information about the key to show that it has now
                // been disabled.
                var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
                {
                    KeyId = keyId,
                });
                Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
            }
        }
    }
```

- Para API obtener más información, consulte DisableKeyla AWS SDK for .NET APIReferencia.

`EnableKey`

En el siguiente ejemplo de código, se muestra cómo usar `EnableKey`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Enable an AWS Key Management Service (AWS KMS) key.
    /// </summary>
    public class EnableKey
    {
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();

            // The identifier of the AWS KMS key to enable. You can use the
            // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

            var request = new EnableKeyRequest
            {
                KeyId = keyId,
            };

            var response = await client.EnableKeyAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                // Retrieve information about the key to show that it has now
                // been enabled.
                var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
                {
                    KeyId = keyId,
                });
```

```
                Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
            }
        }
    }
```

- Para API obtener más información, consulte EnableKeyla AWS SDK for .NET APIReferencia.

`ListAliases`

En el siguiente ejemplo de código, se muestra cómo usar `ListAliases`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// List the AWS Key Management Service (AWS KMS) aliases that have been defined
for
    /// the keys in the same AWS Region as the default user. If you want to list
    /// the aliases in a different Region, pass the Region to the client
    /// constructor.
    /// </summary>
    public class ListAliases
    {
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();
            var request = new ListAliasesRequest();
            var response = new ListAliasesResponse();
```

```
            do
            {
                response = await client.ListAliasesAsync(request);

                response.Aliases.ForEach(alias =>
                {
                    Console.WriteLine($"Created: {alias.CreationDate} Last Update:
{alias.LastUpdatedDate} Name: {alias.AliasName}");
                });

                request.Marker = response.NextMarker;
            }
            while (response.Truncated);
        }
    }
```

- Para API obtener más información, consulte ListAliasesla AWS SDK for .NET APIReferencia.

`ListGrants`

En el siguiente ejemplo de código, se muestra cómo usar `ListGrants`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// List the AWS Key Management Service (AWS KMS) grants that are associated
with
```

```
    /// a specific key.
    /// </summary>
    public class ListGrants
    {
        public static async Task Main()
        {
            // The identifier of the AWS KMS key to disable. You can use the
            // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";
            var client = new AmazonKeyManagementServiceClient();
            var request = new ListGrantsRequest
            {
                KeyId = keyId,
            };

            var response = new ListGrantsResponse();

            do
            {
                response = await client.ListGrantsAsync(request);

                response.Grants.ForEach(grant =>
                {
                    Console.WriteLine($"{grant.GrantId}");
                });

                request.Marker = response.NextMarker;
            }
            while (response.Truncated);
        }
    }
```

- Para API obtener más información, consulte ListGrantsla AWS SDK for .NET APIReferencia.

`ListKeys`

En el siguiente ejemplo de código, se muestra cómo usar `ListKeys`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// List the AWS Key Managements Service (AWS KMS) keys for the AWS Region
/// of the default user. To list keys in another AWS Region, supply the Region
/// as a parameter to the client constructor.
/// </summary>
public class ListKeys
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();
        var request = new ListKeysRequest();
        var response = new ListKeysResponse();

        do
        {
            response = await client.ListKeysAsync(request);

            response.Keys.ForEach(key =>
            {
                Console.WriteLine($"ID: {key.KeyId}, {key.KeyArn}");
            });

            // Set the Marker property when response.Truncated is true
            // in order to get the next keys.
            request.Marker = response.NextMarker;
        }
        while (response.Truncated);
    }
}
```

- Para API obtener más información, consulte ListKeysla AWS SDK for .NET APIReferencia.

Ejemplos de Lambda que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Lambda.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Lambda

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Lambda.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
```

```
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
 existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {
            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Para API obtener más información, consulte ListFunctionsla AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios
- Ejemplos sin servidor

Acciones

`CreateFunction`

En el siguiente ejemplo de código, se muestra cómo usar `CreateFunction`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
```

```
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }
```

- Para API obtener más información, consulte CreateFunctionla AWS SDK for .NET APIReferencia.

`DeleteFunction`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteFunction`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}
```

- Para API obtener más información, consulte DeleteFunctionla AWS SDK for .NET APIReferencia.

`GetFunction`

En el siguiente ejemplo de código, se muestra cómo usar `GetFunction`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}
```

- Para API obtener más información, consulte GetFunctionla AWS SDK for .NET APIReferencia.

`Invoke`

En el siguiente ejemplo de código, se muestra cómo usar `Invoke`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

- Para API obtener más información, consulte Invoke in AWS SDK for .NET APIReference.

`ListFunctions`

En el siguiente ejemplo de código, se muestra cómo usar `ListFunctions`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. 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 Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Para API obtener más información, consulte ListFunctionsla AWS SDK for .NET APIReferencia.

`UpdateFunctionCode`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateFunctionCode`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
 {response.LastModified}.");
    }
```

- Para API obtener más información, consulte UpdateFunctionCodela AWS SDK for .NET APIReferencia.

`UpdateFunctionConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateFunctionConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
 actions.</param>
    /// <param name="environmentVariables">A dictionary of environment variables.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
 environmentVariables },
        };

        var response = await
 _lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte UpdateFunctionConfigurationla AWS SDK for .NET APIReferencia.

Escenarios

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- APIPuerta de enlace
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Creación de una aplicación para analizar los comentarios de los clientes

El siguiente ejemplo de código muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Comenzar a usar las funciones

En el siguiente ejemplo de código, se muestra cómo:

- Cree un IAM rol y una función Lambda y, a continuación, cargue el código del controlador.
- Invocar la función con un único parámetro y obtener resultados.
- Actualizar el código de la función y configurar con una variable de entorno.
- Invocar la función con un nuevo parámetro y obtener resultados. Mostrar el registro de ejecución devuelto.
- Enumerar las funciones de su cuenta y, luego, limpiar los recursos.

Para obtener información, consulte Crear una función de Lambda con la consola.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Cree métodos que realicen acciones de Lambda.

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
```

```
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }


    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }
```

```
    /// <summary>
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }


    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

```
    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }


    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };
```

```
        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
 {response.LastModified}.");
    }


    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
 actions.</param>
    /// <param name="environmentVariables">A dictionary of environment variables.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
 environmentVariables },
        };

        var response = await
 _lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


}
```

Cree una función que ejecute el escenario.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonLambda>()
            .AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<LambdaWrapper>()
            .AddTransient<LambdaRoleWrapper>()
            .AddTransient<UIWrapper>()
        )
        .Build();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
```

```
            true) // Optionally load local settings.
        .Build();


        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LambdaBasics>();

        var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
        var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        string functionName = configuration["FunctionName"]!;
        string roleName = configuration["RoleName"]!;
        string policyDocument = "{" +
            " \"Version\": \"2012-10-17\"," +
            " \"Statement\": [ " +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Principal\": {" +
            "            \"Service\": \"lambda.amazonaws.com\" " +
            "    }," +
            "        \"Action\": \"sts:AssumeRole\" " +
            "    }" +
            "]" +
        "}";

        var incrementHandler = configuration["IncrementHandler"];
        var calculatorHandler = configuration["CalculatorHandler"];
        var bucketName = configuration["BucketName"];
        var incrementKey = configuration["IncrementKey"];
        var calculatorKey = configuration["CalculatorKey"];
        var policyArn = configuration["PolicyArn"];

        uiWrapper.DisplayLambdaBasicsOverview();

        // Create the policy to use with the AWS Lambda functions and then attach
the
        // policy to a new role.
        var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

        Console.WriteLine("Waiting for role to become active.");
```

```
        uiWrapper.WaitABit(15, "Wait until the role is active before trying to use
it.");

        // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
        var success = await lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn,
roleName);
        uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to the
role.");

        // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
        // (Amazon S3) bucket.
        uiWrapper.DisplayTitle("Create Lambda Function");
        Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
        var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
            functionName,
            bucketName,
            incrementKey,
            roleArn,
            incrementHandler);

        Console.WriteLine("Waiting for the new function to be available.");
        Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

        // Get the Lambda function.
        Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
        FunctionConfiguration config;
        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.State != State.Active);

        Console.WriteLine($"\nThe function, {functionName} has been created.");
        Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

        uiWrapper.PressEnter();

        // List the Lambda functions.
        uiWrapper.DisplayTitle("Listing all Lambda functions.");
        var functions = await lambdaWrapper.ListFunctionsAsync();
```

```
        DisplayFunctionList(functions);

        uiWrapper.DisplayTitle("Invoke increment function");
        Console.WriteLine("Now that it has been created, invoke the Lambda increment
function.");
        string? value;
        do
        {
            Console.Write("Enter a value to increment: ");
            value = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(value));

        string functionParameters = "{" +
            "\"action\": \"increment\", " +
            "\"x\": \"" + value + "\"" +
        "}";
        var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"{value} + 1 = {answer}.");

        uiWrapper.DisplayTitle("Update function");
        Console.WriteLine("Now update the Lambda function code.");
        await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,
            new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);
```

```
        uiWrapper.DisplayTitle("Call updated function");
        Console.WriteLine("Now call the updated function...");

        bool done = false;

        do
        {
            string? opSelected;

            Console.WriteLine("Select the operation to perform:");
            Console.WriteLine("\t1. add");
            Console.WriteLine("\t2. subtract");
            Console.WriteLine("\t3. multiply");
            Console.WriteLine("\t4. divide");
            Console.WriteLine("\tOr enter \"q\" to quit.");
            Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the operation
you want to perform: ");
            do
            {
                Console.Write("Your choice? ");
                opSelected = Console.ReadLine();
            }
            while (opSelected == string.Empty);

            var operation = (opSelected) switch
            {
                "1" => "add",
                "2" => "subtract",
                "3" => "multiply",
                "4" => "divide",
                "q" => "quit",
                _ => "add",
            };

            if (operation == "quit")
            {
                done = true;
            }
            else
            {
                // Get two numbers and an action from the user.
                value = string.Empty;
                do
```

```
                {
                    Console.Write("Enter the first value: ");
                    value = Console.ReadLine();
                }
                while (value == string.Empty);

                string? value2;
                do
                {
                    Console.Write("Enter a second value: ");
                    value2 = Console.ReadLine();
                }
                while (value2 == string.Empty);

                functionParameters = "{" +
                    "\"action\": \"" + operation + "\", " +
                    "\"x\": \"" + value + "\"," +
                    "\"y\": \"" + value2 + "\"" +
                "}";

                answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
                Console.WriteLine($"The answer when we {operation} the two numbers
is: {answer}.");
            }

            uiWrapper.PressEnter();
        } while (!done);

        // Delete the function created earlier.

        uiWrapper.DisplayTitle("Clean up resources");
        // Detach the IAM policy from the IAM role.
        Console.WriteLine("First detach the IAM policy from the role.");
        success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
        uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached from
the role.");

        Console.WriteLine("Delete the AWS Lambda function.");
        success = await lambdaWrapper.DeleteFunctionAsync(functionName);
        if (success)
        {
            Console.WriteLine($"The {functionName} function was deleted.");
```

```
        }
        else
        {
            Console.WriteLine($"Could not remove the function {functionName}");
        }

        // Now delete the IAM role created for use with the functions
        // created by the application.
        Console.WriteLine("Now we can delete the role that we created.");
        success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
        if (success)
        {
            Console.WriteLine("The role has been successfully removed.");
        }
        else
        {
            Console.WriteLine("Couldn't delete the role.");
        }

        Console.WriteLine("The Lambda Scenario is now complete.");
        uiWrapper.PressEnter();

        // Displays a formatted list of existing functions returned by the
        // LambdaMethods.ListFunctions.
        void DisplayFunctionList(List<FunctionConfiguration> functions)
        {
            functions.ForEach(functionConfig =>
            {

 Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
            });
        }
    }
}


namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;
```

```
    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
 to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
 policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
```

```
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}


namespace LambdaScenarioCommon;
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM)
 role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
 permissions.");
```

```
        Console.WriteLine("\t3. Creates a Lambda function that increments the value
passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
```

```
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Defina un controlador de Lambda que aumente un número.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaIncrement;

public class Function
{

    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
```

```
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}
```

Defina un segundo controlador de Lambda que realice operaciones aritméticas.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaCalculator;

public class Function
{

    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
```

```
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
                else
                    result = x / y;
                break;
            default:
                Console.Error.WriteLine($"{action} is not a valid operation.");
                result = 0;
                break;
        }
        return result;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateFunction

- DeleteFunction
- GetFunction
- Invoke
- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Transformación de datos con S3 Object Lambda

En el siguiente ejemplo de código se muestra cómo transformar datos para su aplicación con S3 Object Lambda.

AWS SDK for .NET

Muestra cómo añadir código personalizado a GET las solicitudes estándar de S3 para modificar el objeto solicitado recuperado de S3 de forma que se adapte a las necesidades del cliente o la aplicación solicitante.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Lambda
- Amazon S3

Ejemplos sin servidor

Invocar una función de Lambda desde un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al recibir registros de un flujo de Kinesis. La función recupera la carga útil de Kinesis, la decodifica desde Base64 y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de Kinesis con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
```

```
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Invocación de una función de Lambda desde un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una transmisión de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de DynamoDB con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Invocar una función Lambda desde un disparador de Amazon MSK

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de registros de un clúster de AmazonMSK. La función recupera la MSK carga útil y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un MSK evento de Amazon con Lambda usando. NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to process.</
param>
    /// <param name="context">The ILambdaContext that provides methods for logging
 and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
```

```
            }
        }
    }


}
```

Invocación de una función de Lambda desde un desencadenador de Amazon S3

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al cargar un objeto en un bucket de S3. La función recupera el nombre del bucket de S3 y la clave del objeto del parámetro de evento y llama a Amazon S3 API para recuperar y registrar el tipo de contenido del objeto.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de S3 con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace S3Integration
{
```

```
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Invocar una función Lambda desde un disparador de Amazon SNS

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de mensajes de un SNS tema. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un SNS evento con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
 context)
```

```
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Invocar una función Lambda desde un disparador de Amazon SQS

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de mensajes de una SQS cola. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un SQS evento con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis

En el siguiente ejemplo de código se muestra cómo implementar una respuesta por lotes parcial para funciones de Lambda que reciben eventos de un flujo de Kinesis. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de errores en los artículos de lote de Kinesis con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
```

```
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
immediately.
                   Lambda will immediately begin to retry processing from this
failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
```

```
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una transmisión de DynamoDB. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de errores en los elementos de lote de DynamoDB con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Notificación de errores de artículos de lotes para funciones de Lambda con un activador de Amazon SQS

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una SQS cola. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de los errores de los elementos del SQS lote con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
```

```
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

MediaConvert ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with MediaConvert.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola MediaConvert

En el siguiente ejemplo de código se muestra cómo empezar a utilizar AWS Elemental MediaConvert.

AWS SDK for .NET

> **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.MediaConvert;
using Amazon.MediaConvert.Model;

namespace MediaConvertActions;

public static class HelloMediaConvert
{
    static async Task Main(string[] args)
    {
        // Create the client using the default profile.
        var mediaConvertClient = new AmazonMediaConvertClient();

        Console.WriteLine($"Hello AWS Elemental MediaConvert! Your MediaConvert Jobs
 are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get some MediaConvert jobs.
        var response = await mediaConvertClient.ListJobsAsync(
            new ListJobsRequest()
            {
                MaxResults = 10
            }
            );

        foreach (var job in response.Jobs)
        {
            Console.WriteLine($"\tJob: {job.Id} status {job.Status}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte DescribeEndpointsla AWS SDK for .NET APIReferencia.

Temas

- Acciones

Acciones

`CreateJob`

En el siguiente ejemplo de código, se muestra cómo usar `CreateJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Configure las ubicaciones de los archivos, el cliente y el contenedor.

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
 settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

```
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"Creating job for input file {fileInput}.");
        var jobId = await wrapper.CreateJob(mediaConvertRole!, fileInput!,
 fileOutput!);
        Console.WriteLine($"Created job with Job ID: {jobId}");
        Console.WriteLine(new string('-', 80));
```

Cree el trabajo mediante el método de contenedor y devuelva el ID del trabajo.

```
    /// <summary>
    /// Create a job to convert a media file.
    /// </summary>
    /// <param name="mediaConvertRole">The Amazon Resource Name (ARN) of the media
 convert role, as specified here:
    /// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-in-
mediaconvert-configured.html</param>
    /// <param name="fileInput">The Amazon Simple Storage Service (Amazon S3)
 location of the input media file.</param>
    /// <param name="fileOutput">The Amazon S3 location for the output media file.</
param>
    /// <returns>The ID of the new job.</returns>
    public async Task<string> CreateJob(string mediaConvertRole, string fileInput,
        string fileOutput)
    {
        CreateJobRequest createJobRequest = new CreateJobRequest
        {
            Role = mediaConvertRole
        };

        createJobRequest.UserMetadata.Add("Customer", "Amazon");

        JobSettings jobSettings = new JobSettings
        {
            AdAvailOffset = 0,
            TimecodeConfig = new TimecodeConfig
            {
                Source = TimecodeSource.EMBEDDED
            }
        };
        createJobRequest.Settings = jobSettings;

        #region OutputGroup
```

```
        OutputGroup ofg = new OutputGroup
        {
            Name = "File Group",
            OutputGroupSettings = new OutputGroupSettings
            {
                Type = OutputGroupType.FILE_GROUP_SETTINGS,
                FileGroupSettings = new FileGroupSettings
                {
                    Destination = fileOutput
                }
            }
        };

        Output output = new Output
        {
            NameModifier = "_1"
        };

        #region VideoDescription

        VideoDescription vdes = new VideoDescription
        {
            ScalingBehavior = ScalingBehavior.DEFAULT,
            TimecodeInsertion = VideoTimecodeInsertion.DISABLED,
            AntiAlias = AntiAlias.ENABLED,
            Sharpness = 50,
            AfdSignaling = AfdSignaling.NONE,
            DropFrameTimecode = DropFrameTimecode.ENABLED,
            RespondToAfd = RespondToAfd.NONE,
            ColorMetadata = ColorMetadata.INSERT,
            CodecSettings = new VideoCodecSettings
            {
                Codec = VideoCodec.H_264
            }
        };
        output.VideoDescription = vdes;

        H264Settings h264 = new H264Settings
        {
            InterlaceMode = H264InterlaceMode.PROGRESSIVE,
            NumberReferenceFrames = 3,
            Syntax = H264Syntax.DEFAULT,
            Softness = 0,
```

```
            GopClosedCadence = 1,
            GopSize = 90,
            Slices = 1,
            GopBReference = H264GopBReference.DISABLED,
            SlowPal = H264SlowPal.DISABLED,
            SpatialAdaptiveQuantization = H264SpatialAdaptiveQuantization.ENABLED,
            TemporalAdaptiveQuantization = H264TemporalAdaptiveQuantization.ENABLED,
            FlickerAdaptiveQuantization = H264FlickerAdaptiveQuantization.DISABLED,
            EntropyEncoding = H264EntropyEncoding.CABAC,
            Bitrate = 5000000,
            FramerateControl = H264FramerateControl.SPECIFIED,
            RateControlMode = H264RateControlMode.CBR,
            CodecProfile = H264CodecProfile.MAIN,
            Telecine = H264Telecine.NONE,
            MinIInterval = 0,
            AdaptiveQuantization = H264AdaptiveQuantization.HIGH,
            CodecLevel = H264CodecLevel.AUTO,
            FieldEncoding = H264FieldEncoding.PAFF,
            SceneChangeDetect = H264SceneChangeDetect.ENABLED,
            QualityTuningLevel = H264QualityTuningLevel.SINGLE_PASS,
            FramerateConversionAlgorithm =
                H264FramerateConversionAlgorithm.DUPLICATE_DROP,
            UnregisteredSeiTimecode = H264UnregisteredSeiTimecode.DISABLED,
            GopSizeUnits = H264GopSizeUnits.FRAMES,
            ParControl = H264ParControl.SPECIFIED,
            NumberBFramesBetweenReferenceFrames = 2,
            RepeatPps = H264RepeatPps.DISABLED,
            FramerateNumerator = 30,
            FramerateDenominator = 1,
            ParNumerator = 1,
            ParDenominator = 1
        };
        output.VideoDescription.CodecSettings.H264Settings = h264;

        #endregion VideoDescription

        #region AudioDescription

        AudioDescription ades = new AudioDescription
        {
            LanguageCodeControl = AudioLanguageCodeControl.FOLLOW_INPUT,
            // This name matches one specified in the following Inputs.
            AudioSourceName = "Audio Selector 1",
            CodecSettings = new AudioCodecSettings
```

```
            {
                Codec = AudioCodec.AAC
            }
        };

        AacSettings aac = new AacSettings
        {
            AudioDescriptionBroadcasterMix =
AacAudioDescriptionBroadcasterMix.NORMAL,
            RateControlMode = AacRateControlMode.CBR,
            CodecProfile = AacCodecProfile.LC,
            CodingMode = AacCodingMode.CODING_MODE_2_0,
            RawFormat = AacRawFormat.NONE,
            SampleRate = 48000,
            Specification = AacSpecification.MPEG4,
            Bitrate = 64000
        };
        ades.CodecSettings.AacSettings = aac;
        output.AudioDescriptions.Add(ades);

        #endregion AudioDescription

        #region Mp4 Container

        output.ContainerSettings = new ContainerSettings
        {
            Container = ContainerType.MP4
        };
        Mp4Settings mp4 = new Mp4Settings
        {
            CslgAtom = Mp4CslgAtom.INCLUDE,
            FreeSpaceBox = Mp4FreeSpaceBox.EXCLUDE,
            MoovPlacement = Mp4MoovPlacement.PROGRESSIVE_DOWNLOAD
        };
        output.ContainerSettings.Mp4Settings = mp4;

        #endregion Mp4 Container

        ofg.Outputs.Add(output);
        createJobRequest.Settings.OutputGroups.Add(ofg);

        #endregion OutputGroup

        #region Input
```

```
        Input input = new Input
        {
            FilterEnable = InputFilterEnable.AUTO,
            PsiControl = InputPsiControl.USE_PSI,
            FilterStrength = 0,
            DeblockFilter = InputDeblockFilter.DISABLED,
            DenoiseFilter = InputDenoiseFilter.DISABLED,
            TimecodeSource = InputTimecodeSource.EMBEDDED,
            FileInput = fileInput
        };

        AudioSelector audsel = new AudioSelector
        {
            Offset = 0,
            DefaultSelection = AudioDefaultSelection.NOT_DEFAULT,
            ProgramSelection = 1,
            SelectorType = AudioSelectorType.TRACK
        };
        audsel.Tracks.Add(1);
        input.AudioSelectors.Add("Audio Selector 1", audsel);

        input.VideoSelector = new VideoSelector
        {
            ColorSpace = ColorSpace.FOLLOW
        };

        createJobRequest.Settings.Inputs.Add(input);

        #endregion Input

        CreateJobResponse createJobResponse =
            await _amazonMediaConvert.CreateJobAsync(createJobRequest);

        var jobId = createJobResponse.Job.Id;

        return jobId;
    }
```

- Para API obtener más información, consulte CreateJobla AWS SDK for .NET APIReferencia.

`GetJob`

En el siguiente ejemplo de código, se muestra cómo usar `GetJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Configure las ubicaciones de los archivos, el cliente y el contenedor.

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
 settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

Obtenga un trabajo por su ID.

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Getting job information for Job ID {jobId}");
        var job = await wrapper.GetJobById(jobId);
        Console.WriteLine($"Job {job.Id} created on {job.CreatedAt:d} has status
 {job.Status}.");
        Console.WriteLine(new string('-', 80));
```

```
    /// <summary>
```

```
    /// Get the job information for a job by its ID.
    /// </summary>
    /// <param name="jobId">The ID of the job.</param>
    /// <returns>The Job object.</returns>
    public async Task<Job> GetJobById(string jobId)
    {
        var jobResponse = await _amazonMediaConvert.GetJobAsync(
                new GetJobRequest
                {
                    Id = jobId
                });

        return jobResponse.Job;
    }
```

- Para API obtener más información, consulte GetJobla AWS SDK for .NET APIReferencia.

`ListJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Configure las ubicaciones de los archivos, el cliente y el contenedor.

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
 settings.local.json.
```

```
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

Enumerar los trabajos que tengan un estado determinado.

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Listing all complete jobs.");
        var completeJobs = await wrapper.ListAllJobsByStatus(JobStatus.COMPLETE);
        completeJobs.ForEach(j =>
        {
            Console.WriteLine($"Job {j.Id} created on {j.CreatedAt:d} has status
{j.Status}.");
        });
```

Enumerar los trabajos mediante un paginador.

```
    /// <summary>
    /// List all of the jobs with a particular status using a paginator.
    /// </summary>
    /// <param name="status">The status to use when listing jobs.</param>
    /// <returns>The list of jobs matching the status.</returns>
    public async Task<List<Job>> ListAllJobsByStatus(JobStatus? status = null)
    {
        var returnedJobs = new List<Job>();

        var paginatedJobs = _amazonMediaConvert.Paginators.ListJobs(
                new ListJobsRequest
                {
                    Status = status
                });

        // Get the entire list using the paginator.
        await foreach (var job in paginatedJobs.Jobs)
        {
            returnedJobs.Add(job);
        }
```

```
        return returnedJobs;
    }
```

- Para API obtener más información, consulte ListJobsla AWS SDK for .NET APIReferencia.

MSKEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonMSK.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Ejemplos sin servidor

Ejemplos sin servidor

Invocar una función Lambda desde un disparador de Amazon MSK

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de registros de un clúster de AmazonMSK. La función recupera la MSK carga útil y registra el contenido del registro.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un MSK evento de Amazon con Lambda usando. NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to process.</
param>
    /// <param name="context">The ILambdaContext that provides methods for logging
 and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }


}
```

Ejemplos de organizaciones que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de with AWS SDK for .NET Organizations.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`AttachPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `AttachPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to attach an AWS Organizations policy to an organization,
/// an organizational unit, or an account.
/// </summary>
public class AttachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then calls the
    /// AttachPolicyAsync method to attach the policy to the root
    /// organization.
    /// </summary>
    public static async Task Main()
```

```
        {
            IAmazonOrganizations client = new AmazonOrganizationsClient();
            var policyId = "p-00000000";
            var targetId = "r-0000";

            var request = new AttachPolicyRequest
            {
                PolicyId = policyId,
                TargetId = targetId,
            };

            var response = await client.AttachPolicyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully attached Policy ID {policyId} to
Target ID: {targetId}.");
            }
            else
            {
                Console.WriteLine("Was not successful in attaching the policy.");
            }
        }
    }
```

- Para API obtener más información, consulte AttachPolicyla AWS SDK for .NET APIReferencia.

`CreateAccount`

En el siguiente ejemplo de código, se muestra cómo usar `CreateAccount`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
```

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

    /// <summary>
    /// Creates a new AWS Organizations account.
    /// </summary>
    public class CreateAccount
    {
        /// <summary>
        /// Initializes an Organizations client object and uses it to create
        /// the new account with the name specified in accountName.
        /// </summary>
        public static async Task Main()
        {
            IAmazonOrganizations client = new AmazonOrganizationsClient();
            var accountName = "ExampleAccount";
            var email = "someone@example.com";

            var request = new CreateAccountRequest
            {
                AccountName = accountName,
                Email = email,
            };

            var response = await client.CreateAccountAsync(request);
            var status = response.CreateAccountStatus;

            Console.WriteLine($"The staus of {status.AccountName} is
{status.State}.");
        }
    }
```

- Para API obtener más información, consulte CreateAccountla AWS SDK for .NET APIReferencia.

`CreateOrganization`

En el siguiente ejemplo de código, se muestra cómo usar `CreateOrganization`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Creates an organization in AWS Organizations.
/// </summary>
public class CreateOrganization
{
    /// <summary>
    /// Creates an Organizations client object and then uses it to create
    /// a new organization with the default user as the administrator, and
    /// then displays information about the new organization.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.CreateOrganizationAsync(new
CreateOrganizationRequest
        {
            FeatureSet = "ALL",
        });

        Organization newOrg = response.Organization;

        Console.WriteLine($"Organization: {newOrg.Id} Main Accoount:
{newOrg.MasterAccountId}");
    }
}
```

- Para API obtener más información, consulte CreateOrganizationla AWS SDK for .NET APIReferencia.

`CreateOrganizationalUnit`

En el siguiente ejemplo de código, se muestra cómo usar `CreateOrganizationalUnit`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Creates a new organizational unit in AWS Organizations.
/// </summary>
public class CreateOrganizationalUnit
{
    /// <summary>
    /// Initializes an Organizations client object and then uses it to call
    /// the CreateOrganizationalUnit method. If the call succeeds, it
    /// displays information about the new organizational unit.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitName = "ProductDevelopmentUnit";

        var request = new CreateOrganizationalUnitRequest
        {
            Name = orgUnitName,
            ParentId = "r-0000",
        };
```

```
            var response = await client.CreateOrganizationalUnitAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully created organizational unit:
{orgUnitName}.");
                Console.WriteLine($"Organizational unit {orgUnitName} Details");
                Console.WriteLine($"ARN: {response.OrganizationalUnit.Arn} Id:
{response.OrganizationalUnit.Id}");
            }
            else
            {
                Console.WriteLine("Could not create new organizational unit.");
            }
        }
    }
```

- Para API obtener más información, consulte CreateOrganizationalUnitla AWS SDK for .NET APIReferencia.

`CreatePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `CreatePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
```

```
    /// Creates a new AWS Organizations Policy.
    /// </summary>
    public class CreatePolicy
    {
        /// <summary>
        /// Initializes the AWS Organizations client object, uses it to
        /// create a new Organizations Policy, and then displays information
        /// about the newly created Policy.
        /// </summary>
        public static async Task Main()
        {
            IAmazonOrganizations client = new AmazonOrganizationsClient();
            var policyContent = "{" +
                "   \"Version\": \"2012-10-17\"," +
                " \"Statement\" : [{" +
                    " \"Action\" : [\"s3:*\"]," +
                    " \"Effect\" : \"Allow\"," +
                    " \"Resource\" : \"*\"" +
                "}]" +
            "}";

            try
            {
                var response = await client.CreatePolicyAsync(new
CreatePolicyRequest
                {
                    Content = policyContent,
                    Description = "Enables admins of attached accounts to delegate
all Amazon S3 permissions",
                    Name = "AllowAllS3Actions",
                    Type = "SERVICE_CONTROL_POLICY",
                });

                Policy policy = response.Policy;
                Console.WriteLine($"{policy.PolicySummary.Name} has the following
content: {policy.Content}");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- Para API obtener más información, consulte CreatePolicyla AWS SDK for .NET APIReferencia.

`DeleteOrganization`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteOrganization`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to delete an existing organization using the AWS
    /// Organizations Service.
    /// </summary>
    public class DeleteOrganization
    {
        /// <summary>
        /// Initializes the Organizations client and then calls
        /// DeleteOrganizationAsync to delete the organization.
        /// </summary>
        public static async Task Main()
        {
            // Create the client object using the default account.
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var response = await client.DeleteOrganizationAsync(new
DeleteOrganizationRequest());

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine("Successfully deleted organization.");
```

```
            }
            else
            {
                Console.WriteLine("Could not delete organization.");
            }
        }
    }
```

- Para API obtener más información, consulte DeleteOrganizationla AWS SDK for .NET APIReferencia.

`DeleteOrganizationalUnit`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteOrganizationalUnit`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to delete an existing AWS Organizations organizational unit.
/// </summary>
public class DeleteOrganizationalUnit
{
    /// <summary>
    /// Initializes the Organizations client object and calls
    /// DeleteOrganizationalUnitAsync to delete the organizational unit
    /// with the selected ID.
    /// </summary>
    public static async Task Main()
```

```
        {
            // Create the client object using the default account.
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var orgUnitId = "ou-0000-00000000";

            var request = new DeleteOrganizationalUnitRequest
            {
                OrganizationalUnitId = orgUnitId,
            };

            var response = await client.DeleteOrganizationalUnitAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully deleted the organizational unit
with ID: {orgUnitId}.");
            }
            else
            {
                Console.WriteLine($"Could not delete the organizational unit with
ID: {orgUnitId}.");
            }
        }
    }
```

- Para API obtener más información, consulte DeleteOrganizationalUnitla AWS SDK for .NET APIReferencia.

`DeletePolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DeletePolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Deletes an existing AWS Organizations policy.
/// </summary>
public class DeletePolicy
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// delete the policy with the specified policyId.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-00000000";

        var request = new DeletePolicyRequest
        {
            PolicyId = policyId,
        };

        var response = await client.DeletePolicyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted Policy: {policyId}.");
        }
        else
        {
            Console.WriteLine($"Could not delete Policy: {policyId}.");
        }
    }
}
```

- Para API obtener más información, consulte DeletePolicyla AWS SDK for .NET APIReferencia.

`DetachPolicy`

En el siguiente ejemplo de código, se muestra cómo usar `DetachPolicy`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to detach a policy from an AWS Organizations organization,
/// organizational unit, or account.
/// </summary>
public class DetachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and uses it to call
    /// DetachPolicyAsync to detach the policy.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-00000000";
        var targetId = "r-0000";

        var request = new DetachPolicyRequest
        {
            PolicyId = policyId,
            TargetId = targetId,
        };

        var response = await client.DetachPolicyAsync(request);
```

```
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully detached policy with Policy Id:
{policyId}.");
            }
            else
            {
                Console.WriteLine("Could not detach the policy.");
            }
        }
    }
```

- Para API obtener más información, consulte DetachPolicyla AWS SDK for .NET APIReferencia.

`ListAccounts`

En el siguiente ejemplo de código, se muestra cómo usar `ListAccounts`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the AWS Organizations service to list the accounts associated
    /// with the default account.
    /// </summary>
    public class ListAccounts
    {
        /// <summary>
        /// Creates the Organizations client and then calls its
        /// ListAccountsAsync method.
```

```
        /// </summary>
        public static async Task Main()
        {
            // Create the client object using the default account.
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var request = new ListAccountsRequest
            {
                MaxResults = 5,
            };

            var response = new ListAccountsResponse();
            try
            {
                do
                {
                    response = await client.ListAccountsAsync(request);
                    response.Accounts.ForEach(a => DisplayAccounts(a));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (AWSOrganizationsNotInUseException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about an Organizations account.
        /// </summary>
        /// <param name="account">An Organizations account for which to display
        /// information on the console.</param>
        private static void DisplayAccounts(Account account)
        {
            string accountInfo = $"{account.Id} {account.Name}\t{account.Status}";

            Console.WriteLine(accountInfo);
        }
    }
```

- Para API obtener más información, consulte ListAccountsla AWS SDK for .NET APIReferencia.

`ListOrganizationalUnitsForParent`

En el siguiente ejemplo de código, se muestra cómo usar `ListOrganizationalUnitsForParent`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Lists the AWS Organizations organizational units that belong to an
/// organization.
/// </summary>
public class ListOrganizationalUnitsForParent
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// call the ListOrganizationalUnitsForParentAsync method to retrieve
    /// the list of organizational units.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var parentId = "r-0000";

        var request = new ListOrganizationalUnitsForParentRequest
```

```
            {
                ParentId = parentId,
                MaxResults = 5,
            };

            var response = new ListOrganizationalUnitsForParentResponse();
            try
            {
                do
                {
                    response = await
client.ListOrganizationalUnitsForParentAsync(request);
                    response.OrganizationalUnits.ForEach(u =>
DisplayOrganizationalUnit(u));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about an Organizations organizational unit.
        /// </summary>
        /// <param name="unit">The OrganizationalUnit for which to display
        /// information.</param>
        public static void DisplayOrganizationalUnit(OrganizationalUnit unit)
        {
            string accountInfo = $"{unit.Id} {unit.Name}\t{unit.Arn}";

            Console.WriteLine(accountInfo);
        }
    }
```

- Para API obtener más información, consulte ListOrganizationalUnitsForParentla AWS SDK for .NET APIReferencia.

`ListPolicies`

En el siguiente ejemplo de código, se muestra cómo usar `ListPolicies`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to list the AWS Organizations policies associated with an
/// organization.
/// </summary>
public class ListPolicies
{
    /// <summary>
    /// Initializes an Organizations client object, and then calls its
    /// ListPoliciesAsync method.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        // The value for the Filter parameter is required and must must be
        // one of the following:
        //     AISERVICES_OPT_OUT_POLICY
        //     BACKUP_POLICY
        //     SERVICE_CONTROL_POLICY
        //     TAG_POLICY
        var request = new ListPoliciesRequest
```

```
            {
                Filter = "SERVICE_CONTROL_POLICY",
                MaxResults = 5,
            };

            var response = new ListPoliciesResponse();
            try
            {
                do
                {
                    response = await client.ListPoliciesAsync(request);
                    response.Policies.ForEach(p => DisplayPolicies(p));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (AWSOrganizationsNotInUseException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about the Organizations policies associated
        /// with an organization.
        /// </summary>
        /// <param name="policy">An Organizations policy summary to display
        /// information on the console.</param>
        private static void DisplayPolicies(PolicySummary policy)
        {
            string policyInfo = $"{policy.Id} {policy.Name}\t{policy.Description}";

            Console.WriteLine(policyInfo);
        }
    }
```

- Para API obtener más información, consulte ListPoliciesla AWS SDK for .NET APIReferencia.

Ejemplos de Amazon Pinpoint con AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Amazon Pinpoint.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`SendMessages`

En el siguiente ejemplo de código, se muestra cómo usar `SendMessages`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enviar un mensaje de correo electrónico.

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendEmailMessage;

public class SendEmailMainClass
```

```
{
    public static async Task Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load test settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

        // The AWS Region that you want to use to send the email. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The "From" address. This address has to be verified in Amazon Pinpoint
        // in the region you're using to send email.
        string senderAddress = configuration["SenderAddress"]!;

        // The address on the "To" line. If your Amazon Pinpoint account is in
        // the sandbox, this address also has to be verified.
        string toAddress = configuration["ToAddress"]!;

        // The Amazon Pinpoint project/application ID to use when you send this
message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        try
        {
            await SendEmailMessage(region, appId, toAddress, senderAddress);
        }
        catch (Exception ex)
        {
            Console.WriteLine("The message wasn't sent. Error message: " +
ex.Message);
        }
    }

    public static async Task<MessageResponse> SendEmailMessage(
        string region, string appId, string toAddress, string senderAddress)
    {
```

```
        var client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

        // The subject line of the email.
        string subject = "Amazon Pinpoint Email test";

        // The body of the email for recipients whose email clients don't
        // support HTML content.
        string textBody = @"Amazon Pinpoint Email Test (.NET)"
                          + "\n---------------------------------"
                          + "\nThis email was sent using the Amazon Pinpoint API
 using the AWS SDK for .NET.";

        // The body of the email for recipients whose email clients support
        // HTML content.
        string htmlBody = @"<html>"
                          + "\n<head></head>"
                          + "\n<body>"
                          + "\n  <h1>Amazon Pinpoint Email Test (AWS SDK for .NET)</
h1>"
                          + "\n  <p>This email was sent using the "
                          + "\n    <a href='https://aws.amazon.com/pinpoint/'>Amazon
 Pinpoint</a> API "
                          + "\n    using the <a href='https://aws.amazon.com/sdk-
for-net/'>AWS SDK for .NET</a>"
                          + "\n  </p>"
                          + "\n</body>"
                          + "\n</html>";

        // The character encoding the you want to use for the subject line and
        // message body of the email.
        string charset = "UTF-8";

        var sendRequest = new SendMessagesRequest
        {
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses = new Dictionary<string, AddressConfiguration>
                {
                    {
                        toAddress,
                        new AddressConfiguration
                        {
```

```
                            ChannelType = ChannelType.EMAIL
                        }
                    }
                },
                MessageConfiguration = new DirectMessageConfiguration
                {
                    EmailMessage = new EmailMessage
                    {
                        FromAddress = senderAddress,
                        SimpleEmail = new SimpleEmail
                        {
                            HtmlPart = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = htmlBody
                            },
                            TextPart = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = textBody
                            },
                            Subject = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = subject
                            }
                        }
                    }
                }
            }
        };
        Console.WriteLine("Sending message...");
        SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
        Console.WriteLine("Message sent!");
        return response.MessageResponse;
    }
}
```

Envía un SMS mensaje.

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendSmsMessage;

public class SendSmsMessageMainClass
{
    public static async Task Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // The AWS Region that you want to use to send the message. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The phone number or short code to send the message from. The phone number
        // or short code that you specify has to be associated with your Amazon
 Pinpoint
        // account. For best results, specify long codes in E.164 format.
        string originationNumber = configuration["OriginationNumber"]!;

        // The recipient's phone number.  For best results, you should specify the
        // phone number in E.164 format.
        string destinationNumber = configuration["DestinationNumber"]!;

        // The Pinpoint project/ application ID to use when you send this message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        // The type of SMS message that you want to send. If you plan to send
        // time-sensitive content, specify TRANSACTIONAL. If you plan to send
        // marketing-related content, specify PROMOTIONAL.
        MessageType messageType = MessageType.TRANSACTIONAL;

        // The registered keyword associated with the originating short code.
```

```
        string? registeredKeyword = configuration["RegisteredKeyword"];

        // The sender ID to use when sending the message. Support for sender ID
        // varies by country or region. For more information, see
        // https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-
countries.html
        string? senderId = configuration["SenderId"];

        try
        {
            var response = await SendSmsMessage(region, appId, destinationNumber,
                originationNumber, registeredKeyword, senderId, messageType);
            Console.WriteLine($"Message sent to
 {response.MessageResponse.Result.Count} recipient(s).");
            foreach (var messageResultValue in
                     response.MessageResponse.Result.Select(r => r.Value))
            {
                Console.WriteLine($"{messageResultValue.MessageId} Status:
 {messageResultValue.DeliveryStatus}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("The message wasn't sent. Error message: " +
 ex.Message);
        }
    }

    public static async Task<SendMessagesResponse> SendSmsMessage(
        string region, string appId, string destinationNumber, string
 originationNumber,
        string? keyword, string? senderId, MessageType messageType)
    {

        // The content of the SMS message.
        string message = "This message was sent through Amazon Pinpoint using" +
                         " the AWS SDK for .NET. Reply STOP to opt out.";


        var client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

        SendMessagesRequest sendRequest = new SendMessagesRequest
        {
```

```
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses =
                    new Dictionary<string, AddressConfiguration>
                    {
                        {
                            destinationNumber,
                            new AddressConfiguration { ChannelType =
 ChannelType.SMS }
                        }
                    },
                MessageConfiguration = new DirectMessageConfiguration
                {
                    SMSMessage = new SMSMessage
                    {
                        Body = message,
                        MessageType = MessageType.TRANSACTIONAL,
                        OriginationNumber = originationNumber,
                        SenderId = senderId,
                        Keyword = keyword
                    }
                }
            }
        };
        SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
        return response;
    }
}
```

- Para API obtener más información, consulte SendMessagesla AWS SDK for .NET APIReferencia.

Ejemplos de Amazon Polly que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Amazon Polly.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`DeleteLexicon`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteLexicon`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Deletes an existing Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class DeleteLexicon
{
    public static async Task Main()
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();
```

```
            var success = await DeletePollyLexiconAsync(client, lexiconName);

            if (success)
            {
                Console.WriteLine($"Successfully deleted {lexiconName}.");
            }
            else
            {
                Console.WriteLine($"Could not delete {lexiconName}.");
            }
        }

        /// <summary>
        /// Deletes the named Amazon Polly lexicon.
        /// </summary>
        /// <param name="client">The initialized Amazon Polly client object.</param>
        /// <param name="lexiconName">The name of the Amazon Polly lexicon to
        /// delete.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeletePollyLexiconAsync(
            AmazonPollyClient client,
            string lexiconName)
        {
            var deleteLexiconRequest = new DeleteLexiconRequest()
            {
                Name = lexiconName,
            };

            var response = await client.DeleteLexiconAsync(deleteLexiconRequest);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Para API obtener más información, consulte DeleteLexiconla AWS SDK for .NET APIReferencia.

`DescribeVoices`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeVoices`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

public class DescribeVoices
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();

        var allVoicesRequest = new DescribeVoicesRequest();
        var enUsVoicesRequest = new DescribeVoicesRequest()
        {
            LanguageCode = "en-US",
        };

        try
        {
            string nextToken;
            do
            {
                var allVoicesResponse = await
client.DescribeVoicesAsync(allVoicesRequest);
                nextToken = allVoicesResponse.NextToken;
                allVoicesRequest.NextToken = nextToken;

                Console.WriteLine("\nAll voices: ");
                allVoicesResponse.Voices.ForEach(voice =>
                {
                    DisplayVoiceInfo(voice);
                });
            }
            while (nextToken is not null);
```

```
                do
                {
                    var enUsVoicesResponse = await
client.DescribeVoicesAsync(enUsVoicesRequest);
                    nextToken = enUsVoicesResponse.NextToken;
                    enUsVoicesRequest.NextToken = nextToken;

                    Console.WriteLine("\nen-US voices: ");
                    enUsVoicesResponse.Voices.ForEach(voice =>
                    {
                        DisplayVoiceInfo(voice);
                    });
                }
                while (nextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception caught: " + ex.Message);
            }
        }

        public static void DisplayVoiceInfo(Voice voice)
        {
            Console.WriteLine($" Name: {voice.Name}\tGender:
{voice.Gender}\tLanguageName: {voice.LanguageName}");
        }
    }
```

- Para API obtener más información, consulte DescribeVoicesla AWS SDK for .NET APIReferencia.

`GetLexicon`

En el siguiente ejemplo de código, se muestra cómo usar `GetLexicon`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Retrieves information about a specific Amazon Polly lexicon.
    /// </summary>
    public class GetLexicon
    {
        public static async Task Main(string[] args)
        {
            string lexiconName = "SampleLexicon";

            var client = new AmazonPollyClient();

            await GetPollyLexiconAsync(client, lexiconName);
        }

        public static async Task GetPollyLexiconAsync(AmazonPollyClient client,
string lexiconName)
        {
            var getLexiconRequest = new GetLexiconRequest()
            {
                Name = lexiconName,
            };

            try
            {
                var response = await client.GetLexiconAsync(getLexiconRequest);
                Console.WriteLine($"Lexicon:\n Name: {response.Lexicon.Name}");
                Console.WriteLine($"Content: {response.Lexicon.Content}");
            }
            catch (Exception ex)
```

```
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }
```

- Para API obtener más información, consulte GetLexiconla AWS SDK for .NET APIReferencia.

`ListLexicons`

En el siguiente ejemplo de código, se muestra cómo usar `ListLexicons`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Lists the Amazon Polly lexicons that have been defined. By default,
/// lists the lexicons that are defined in the same AWS Region as the default
/// user. To view Amazon Polly lexicons that are defined in a different AWS
/// Region, supply it as a parameter to the Amazon Polly constructor.
/// </summary>
public class ListLexicons
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        var request = new ListLexiconsRequest();

        try
```

```
            {
                Console.WriteLine("All voices: ");

                do
                {
                    var response = await client.ListLexiconsAsync(request);
                    request.NextToken = response.NextToken;

                    response.Lexicons.ForEach(lexicon =>
                    {
                        var attributes = lexicon.Attributes;
                        Console.WriteLine($"Name: {lexicon.Name}");
                        Console.WriteLine($"\tAlphabet: {attributes.Alphabet}");
                        Console.WriteLine($"\tLanguageCode:
{attributes.LanguageCode}");
                        Console.WriteLine($"\tLastModified:
{attributes.LastModified}");
                        Console.WriteLine($"\tLexemesCount:
{attributes.LexemesCount}");
                        Console.WriteLine($"\tLexiconArn: {attributes.LexiconArn}");
                        Console.WriteLine($"\tSize: {attributes.Size}");
                    });
                }
                while (request.NextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
```

- Para API obtener más información, consulte ListLexiconsla AWS SDK for .NET APIReferencia.

`PutLexicon`

En el siguiente ejemplo de código, se muestra cómo usar `PutLexicon`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Creates a new Amazon Polly lexicon using the AWS SDK for .NET.
    /// </summary>
    public class PutLexicon
    {
        public static async Task Main()
        {
            string lexiconContent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/
pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
                "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-
lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
                "alphabet=\"ipa\" xml:lang=\"en-US\">" +
                "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>" +
                "</lexicon>";
            string lexiconName = "SampleLexicon";

            var client = new AmazonPollyClient();
            var putLexiconRequest = new PutLexiconRequest()
            {
                Name = lexiconName,
                Content = lexiconContent,
            };

            try
            {
                var response = await client.PutLexiconAsync(putLexiconRequest);
                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
```

```
                    Console.WriteLine($"Successfully created Lexicon:
{lexiconName}.");
                }
                else
                {
                    Console.WriteLine($"Could not create Lexicon: {lexiconName}.");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception caught: " + ex.Message);
            }
        }
    }
```

- Para API obtener más información, consulte PutLexiconla AWS SDK for .NET APIReferencia.

`SynthesizeSpeech`

En el siguiente ejemplo de código, se muestra cómo usar `SynthesizeSpeech`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeech
{
    public static async Task Main()
    {
        string outputFileName = "speech.mp3";
```

```
            string text = "Twas brillig, and the slithy toves did gyre and gimbol in
the wabe";

            var client = new AmazonPollyClient();
            var response = await PollySynthesizeSpeech(client, text);

            WriteSpeechToStream(response.AudioStream, outputFileName);
        }

        /// <summary>
        /// Calls the Amazon Polly SynthesizeSpeechAsync method to convert text
        /// to speech.
        /// </summary>
        /// <param name="client">The Amazon Polly client object used to connect
        /// to the Amazon Polly service.</param>
        /// <param name="text">The text to convert to speech.</param>
        /// <returns>A SynthesizeSpeechResponse object that includes an AudioStream
        /// object with the converted text.</returns>
        private static async Task<SynthesizeSpeechResponse>
PollySynthesizeSpeech(IAmazonPolly client, string text)
        {
            var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            {
                OutputFormat = OutputFormat.Mp3,
                VoiceId = VoiceId.Joanna,
                Text = text,
            };

            var synthesizeSpeechResponse =
                await client.SynthesizeSpeechAsync(synthesizeSpeechRequest);

            return synthesizeSpeechResponse;
        }

        /// <summary>
        /// Writes the AudioStream returned from the call to
        /// SynthesizeSpeechAsync to a file in MP3 format.
        /// </summary>
        /// <param name="audioStream">The AudioStream returned from the
        /// call to the SynthesizeSpeechAsync method.</param>
        /// <param name="outputFileName">The full path to the file in which to
        /// save the audio stream.</param>
        private static void WriteSpeechToStream(Stream audioStream, string
outputFileName)
```

```
        {
            var outputStream = new FileStream(
                outputFileName,
                FileMode.Create,
                FileAccess.Write);
            byte[] buffer = new byte[2 * 1024];
            int readBytes;

            while ((readBytes = audioStream.Read(buffer, 0, 2 * 1024)) > 0)
            {
                outputStream.Write(buffer, 0, readBytes);
            }

            // Flushes the buffer to avoid losing the last second or so of
            // the synthesized text.
            outputStream.Flush();
            Console.WriteLine($"Saved {outputFileName} to disk.");
        }
    }
```

Sintetice la voz a partir del texto mediante marcas de voz con Amazon Polly mediante un. AWS SDK

```
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Polly;
    using Amazon.Polly.Model;

    public class SynthesizeSpeechMarks
    {
        public static async Task Main()
        {
            var client = new AmazonPollyClient();
            string outputFileName = "speechMarks.json";

            var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            {
                OutputFormat = OutputFormat.Json,
                SpeechMarkTypes = new List<string>
```

```
                {
                    SpeechMarkType.Viseme,
                    SpeechMarkType.Word,
                },
                VoiceId = VoiceId.Joanna,
                Text = "This is a sample text to be synthesized.",
            };

            try
            {
                using (var outputStream = new FileStream(outputFileName,
FileMode.Create, FileAccess.Write))
                {
                    var synthesizeSpeechResponse = await
client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
                    var buffer = new byte[2 * 1024];
                    int readBytes;

                    var inputStream = synthesizeSpeechResponse.AudioStream;
                    while ((readBytes = inputStream.Read(buffer, 0, 2 * 1024)) > 0)
                    {
                        outputStream.Write(buffer, 0, readBytes);
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
```

- Para API obtener más información, consulte SynthesizeSpeechla referencia.AWS SDK for .NET API

Escenarios

Creación de una aplicación para analizar los comentarios de los clientes

El siguiente ejemplo de código muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

RDSEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonRDS.

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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola Amazon RDS

Los siguientes ejemplos de código muestran cómo empezar a utilizar AmazonRDS.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

namespace RDSActions;

public static class HelloRds
{
    static async Task Main(string[] args)
    {
        var rdsClient = new AmazonRDSClient();

        Console.WriteLine($"Hello Amazon RDS! Following are some of your DB
 instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
```

```
        // Let's get the first twenty DB instances.
        var response = await rdsClient.DescribeDBInstancesAsync(
            new DescribeDBInstancesRequest()
            {
                MaxRecords = 20 // Must be between 20 and 100.
            });

        foreach (var instance in response.DBInstances)
        {
            Console.WriteLine($"\tDB name: {instance.DBName}");
            Console.WriteLine($"\tArn: {instance.DBInstanceArn}");
            Console.WriteLine($"\tIdentifier: {instance.DBInstanceIdentifier}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte D escribeDBInstances en la AWS SDK for .NET APIreferencia.

Temas

- Conceptos básicos
- Acciones
- Escenarios

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Cree un grupo de parámetros de base de datos personalizado y defina los valores de los parámetros.
- Cree una instancia de base de datos que esté configurada para utilizar el grupo de parámetros. La instancia de base de datos también contiene una base de datos.
- Cree una instantánea de la instancia.
- Elimine la instancia y el grupo de parámetros.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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.

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Returns a list of the available DB engine families using the
 DescribeDBEngineVersionsAsync method.
    2.  Selects an engine family and creates a custom DB parameter group using the
 CreateDBParameterGroupAsync method.
    3.  Gets the parameter groups using the DescribeDBParameterGroupsAsync method.
    4.  Gets parameters in the group using the DescribeDBParameters method.
    5.  Parses and displays parameters in the group.
    6.  Modifies both the auto_increment_offset and auto_increment_increment
 parameters
        using the ModifyDBParameterGroupAsync method.
    7.  Gets and displays the updated parameters using the DescribeDBParameters
 method with a source of "user".
    8.  Gets a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Displays and selects from a list of micro instance classes available for the
 selected engine and version.
    10. Creates an RDS DB instance that contains a MySql database and uses the
 parameter group
        using the CreateDBInstanceAsync method.
    11. Waits for DB instance to be ready using the DescribeDBInstancesAsync method.
    12. Prints out the connection endpoint string for the new DB instance.
```

```
   13. Creates a snapshot of the DB instance using the CreateDBSnapshotAsync
method.
   14. Waits for DB snapshot to be ready using the DescribeDBSnapshots method.
   15. Deletes the DB instance using the DeleteDBInstanceAsync method.
   16. Waits for DB instance to be deleted using the DescribeDbInstances method.
   17. Deletes the parameter group using the DeleteDBParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static RDSWrapper rdsWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon RDS service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<RDSWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<RDSInstanceScenario>();

       rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

       Console.WriteLine(sepBar);
       Console.WriteLine(
           "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
       Console.WriteLine(sepBar);

       try
       {
           var parameterGroupFamily = await ChooseParameterGroupFamily();
```

```
            var parameterGroup = await CreateDbParameterGroup(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParameters(parameterGroup.DBParameterGroupName, parameters);

            await DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

            var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

            var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
                instanceChoice.DBInstanceClass, newInstanceIdentifier);
            if (newInstance != null)
            {
                DisplayConnectionString(newInstance);

                await CreateSnapshot(newInstance);

                await DeleteRdsInstance(newInstance);
            }

            await DeleteParameterGroup(parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the RDS DB parameter group family from a list of available options.
```

```
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamily()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

        Console.WriteLine("1. The following is a list of available DB parameter
group families:");
        int i = 1;
        var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
        foreach (var parameterGroupFamily in parameterGroupFamilies)
        {
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
```

```
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\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 ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                int newValue = 0;
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    Int32.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

        Console.WriteLine(sepBar);
    }
```

```
    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe user source parameters in the group.");

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName, "user");


        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }


    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDbEngineVersion(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
```

```
        {
            Console.WriteLine(
                $"\t{i}. Engine: {version.Engine} Version
 {version.EngineVersion}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDbInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
 engineVersion);

        Console.WriteLine($"8. Available micro DB instance classes for engine
 {engine} and version {engineVersion}:");
        int i = 1;

        // Filter to micro instances for this example.
        allowedInstances = allowedInstances
            .Where(i => i.DBInstanceClass.Contains("micro")).ToList();
```

```
        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("9. Select an available DB instance class by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB instance.</
param>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateRdsNewInstance(DBParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string instanceClass, string
 instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"10. Create a new DB instance with identifier
 {instanceIdentifier}.");
```

```
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await rdsWrapper.DescribeDBInstances();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {
            Console.WriteLine("Please enter an admin user name:");
            var username = Console.ReadLine();

            Console.WriteLine("Please enter an admin password:");
            var password = Console.ReadLine();

            newInstance = await rdsWrapper.CreateDBInstance(
                "ExampleInstance",
                instanceIdentifier,
                parameterGroup.DBParameterGroupName,
                engineName,
                engineVersion,
                instanceClass,
                20,
                username,
                password
            );

            // 11. Wait for the DB instance to be ready.

            Console.WriteLine("11. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                instances = await
rdsWrapper.DescribeDBInstances(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
                newInstance = instances.First();
                Thread.Sleep(30000);
```

```
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("12. New DB instance connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
 "
            + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an RDS DB instance.
    /// </summary>
    /// <param name="instance">DB instance to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"13. Creating snapshot from DB instance
 {instance.DBInstanceIdentifier}.");
        var snapshot = await
 rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
 DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
```

```
        while (!isSnapshotReady)
        {
            var snapshots = await
rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
    {
        Console.WriteLine(sepBar);
        // Delete the DB instance.
        Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
        await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);

        // Wait for the DB instance to delete.
        Console.WriteLine($"16. Waiting for the DB instance to delete...");
        bool isInstanceDeleted = false;

        while (!isInstanceDeleted)
        {
            var instance = await rdsWrapper.DescribeDBInstances();
            isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
newInstance.DBInstanceIdentifier);
            Thread.Sleep(30000);
        }

        Console.WriteLine("DB instance deleted.");
        Console.WriteLine(sepBar);
    }
```

```
    /// <summary>
    /// Delete a DB parameter group.
    /// </summary>
    /// <param name="parameterGroup">The parameter group to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteParameterGroup(DBParameterGroup parameterGroup)
    {
        Console.WriteLine(sepBar);
        // Delete the parameter group.
        Console.WriteLine($"17. Delete the DB parameter group
 {parameterGroup.DBParameterGroupName}.");
        await
 rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

        Console.WriteLine(sepBar);
    }
```

Métodos envolventes utilizados por el escenario para las acciones de la instancia de base de datos.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with DB
 instance operations.
/// </summary>
public partial class RDSWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public RDSWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }


    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
```

```
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }



    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

```
   /// <summary>
   /// Returns a list of DB instances.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
   /// <returns>List of DB instances.</returns>
   public async Task<List<DBInstance>> DescribeDBInstances(string
dbInstanceIdentifier = null)
   {
       var results = new List<DBInstance>();
       var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
           new DescribeDBInstancesRequest
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });
       // Get the entire list using the paginator.
       await foreach (var instances in instancesPaginator.DBInstances)
       {
           results.Add(instances);
       }
       return results;
   }



   /// <summary>
   /// Create an RDS DB instance with a particular set of properties. Use the
action DescribeDBInstancesAsync
   /// to determine when the DB instance is ready to use.
   /// </summary>
   /// <param name="dbName">Name for the DB instance.</param>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</param>
   /// <param name="dbEngine">The engine for the DB instance.</param>
   /// <param name="dbEngineVersion">Version for the DB instance.</param>
   /// <param name="instanceClass">Class for the DB instance.</param>
   /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
   /// <param name="adminName">Admin user name.</param>
   /// <param name="adminPassword">Admin user password.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
```

```
        string parameterGroupName, string dbEngine, string dbEngineVersion,
        string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
    {
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBName = dbName,
                DBInstanceIdentifier = dbInstanceIdentifier,
                DBParameterGroupName = parameterGroupName,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass,
                AllocatedStorage = allocatedStorage,
                MasterUsername = adminName,
                MasterUserPassword = adminPassword
            });

        return response.DBInstance;
    }



    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

Métodos envolventes utilizados por el escenario para los grupos de parámetros de base de datos.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 parameter groups.
/// </summary>
public partial class RDSWrapper
{

    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
 = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }



    /// <summary>
    /// Create a new DB parameter group. Use the action
 DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="family">Family of the DB parameter group.</param>
    /// <param name="description">Description of the DB parameter group.</param>
    /// <returns>The new DB parameter group.</returns>
    public async Task<DBParameterGroup> CreateDBParameterGroup(
        string name, string family, string description)
    {
        var response = await _amazonRDS.CreateDBParameterGroupAsync(
            new CreateDBParameterGroupRequest()
            {
```

```
                DBParameterGroupName = name,
                DBParameterGroupFamily = family,
                Description = description
            });
        return response.DBParameterGroup;
    }



    /// <summary>
    /// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }



    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
 group
    /// or be associated with any DB instances.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
```

```
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }



    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
 dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
```

Métodos envolventes utilizados por el escenario para las acciones de la instantánea de base de datos.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 snapshots.
/// </summary>
public partial class RDSWrapper
{
```

```
   /// <summary>
   /// Create a snapshot of a DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
   /// <returns>DB snapshot object.</returns>
   public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
   {
       var response = await _amazonRDS.CreateDBSnapshotAsync(
           new CreateDBSnapshotRequest()
           {
               DBSnapshotIdentifier = snapshotIdentifier,
               DBInstanceIdentifier = dbInstanceIdentifier
           });

       return response.DBSnapshot;
   }



   /// <summary>
   /// Return a list of DB snapshots for a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <returns>List of DB snapshots.</returns>
   public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
   {
       var results = new List<DBSnapshot>();
       var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
           new DescribeDBSnapshotsRequest()
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });

       // Get the entire list using the paginator.
       await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
       {
           results.Add(snapshots);
       }
       return results;
   }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - C reateDBInstance
  - reateDBParameterGrupo C
  - C reateDBSnapshot
  - D eleteDBInstance
  - eleteDBParameterGrupo D
  - escribeDBEngineVersiones en D
  - D escribeDBInstances
  - escribeDBParameterGrupos D
  - D escribeDBParameters
  - D escribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - odifyDBParameterGrupo M

Acciones

`CreateDBInstance`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBInstance`.

AWS SDK for .NET

> **Note**
>
> Hay más en marcha GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create an RDS DB instance with a particular set of properties. Use the
action DescribeDBInstancesAsync
```

```
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbName">Name for the DB instance.</param>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
    /// <param name="adminName">Admin user name.</param>
    /// <param name="adminPassword">Admin user password.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
        string parameterGroupName, string dbEngine, string dbEngineVersion,
        string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
    {
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBName = dbName,
                DBInstanceIdentifier = dbInstanceIdentifier,
                DBParameterGroupName = parameterGroupName,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass,
                AllocatedStorage = allocatedStorage,
                MasterUsername = adminName,
                MasterUserPassword = adminPassword
            });

        return response.DBInstance;
    }
```

- Para API obtener más información, consulte C reateDBInstance en la AWS SDK for .NET APIreferencia.

`CreateDBParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
   /// to determine when the DB parameter group is ready to use.
   /// </summary>
   /// <param name="name">Name of the DB parameter group.</param>
   /// <param name="family">Family of the DB parameter group.</param>
   /// <param name="description">Description of the DB parameter group.</param>
   /// <returns>The new DB parameter group.</returns>
   public async Task<DBParameterGroup> CreateDBParameterGroup(
       string name, string family, string description)
   {
       var response = await _amazonRDS.CreateDBParameterGroupAsync(
           new CreateDBParameterGroupRequest()
           {
               DBParameterGroupName = name,
               DBParameterGroupFamily = family,
               Description = description
           });
       return response.DBParameterGroup;
   }
```

- Para API obtener más información, consulte el reateDBParameterGrupo C en AWS SDK for .NET APIla referencia.

`CreateDBSnapshot`

En el siguiente ejemplo de código, se muestra cómo usar `CreateDBSnapshot`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
   /// <returns>DB snapshot object.</returns>
   public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
   {
       var response = await _amazonRDS.CreateDBSnapshotAsync(
           new CreateDBSnapshotRequest()
           {
               DBSnapshotIdentifier = snapshotIdentifier,
               DBInstanceIdentifier = dbInstanceIdentifier
           });

       return response.DBSnapshot;
   }
```

- Para API obtener más información, consulte C reateDBSnapshot en la AWS SDK for .NET APIreferencia.

`DeleteDBInstance`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDBInstance`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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> DeleteDBInstance(string dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}
```

- Para API obtener más información, consulte D eleteDBInstance en la AWS SDK for .NET APIreferencia.

`DeleteDBParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteDBParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
group
    /// or be associated with any DB instances.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte el eleteDBParameterGrupo D en AWS SDK for .NET APIla referencia.

`DescribeDBEngineVersions`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBEngineVersions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- Para API obtener más información, consulte escribeDBEnginelas versiones D en AWS SDK for .NET APIla referencia.

`DescribeDBInstances`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBInstances`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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>> DescribeDBInstances(string
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- Para API obtener más información, consulte D escribeDBInstances en la AWS SDK for .NET APIreferencia.

`DescribeDBParameterGroups`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBParameterGroups`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
 = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }
```

- Para API obtener más información, consulte escribeDBParameterlos grupos D en AWS SDK for .NET APIla referencia.

`DescribeDBParameters`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBParameters`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
 dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
```

- Para API obtener más información, consulte D escribeDBParameters en la AWS SDK for .NET APIreferencia.

`DescribeDBSnapshots`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeDBSnapshots`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <returns>List of DB snapshots.</returns>
   public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
   {
       var results = new List<DBSnapshot>();
       var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
           new DescribeDBSnapshotsRequest()
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });

       // Get the entire list using the paginator.
       await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
       {
           results.Add(snapshots);
       }
       return results;
   }
```

- Para API obtener más información, consulte D escribeDBSnapshots en la AWS SDK for .NET APIreferencia.

`DescribeOrderableDBInstanceOptions`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeOrderableDBInstanceOptions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
   {
       // Use a paginator to get a list of DB instance options.
       var results = new List<OrderableDBInstanceOption>();
       var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
           new DescribeOrderableDBInstanceOptionsRequest()
           {
               Engine = engine,
               EngineVersion = engineVersion,
           });
       // Get the entire list using the paginator.
       await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
       {
           results.Add(instanceOptions);
       }
       return results;
   }
```

- Para API obtener más información, consulte DescribeOrderableDBInstanceOptionsla AWS SDK for .NET APIReferencia.

`ModifyDBParameterGroup`

En el siguiente ejemplo de código, se muestra cómo usar `ModifyDBParameterGroup`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }
```

- Para API obtener más información, consulte M odifyDBParameter Group en AWS SDK for .NET APIReference.

Escenarios

Crear un rastreador de elementos de trabajo de Aurora Serverless

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos Amazon Aurora Serverless y utilice Amazon Simple Email Service (AmazonSES) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizarla AWS SDK for .NET para crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos de Amazon Aurora y envíe informes por correo electrónico mediante Amazon Simple Email Service (AmazonSES). En este ejemplo, se utiliza una interfaz creada con React.js para interactuar con unRESTful. NETbackend.

- Integre una aplicación web de React con AWS los servicios.
- Muestre, agregue, actualice y elimine elementos en una tabla de Aurora.
- Envía un informe por correo electrónico de los artículos de trabajo filtrados a través de AmazonSES.
- Implemente y gestione recursos de ejemplo con el AWS CloudFormation script incluido.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Ejemplos RDS de Amazon Data Service que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes AWS SDK for .NET mediante Amazon RDS Data Service.

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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Escenarios

Crear un rastreador de elementos de trabajo de Aurora Serverless

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos Amazon Aurora Serverless y utilice Amazon Simple Email Service (AmazonSES) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizarla AWS SDK for .NET para crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos de Amazon Aurora y envíe informes por correo electrónico mediante Amazon Simple Email Service (AmazonSES). En este ejemplo, se utiliza una interfaz creada con React.js para interactuar con unRESTful. NETbackend.

- Integre una aplicación web de React con AWS los servicios.
- Muestre, agregue, actualice y elimine elementos en una tabla de Aurora.
- Envía un informe por correo electrónico de los artículos de trabajo filtrados a través de AmazonSES.
- Implemente y gestione recursos de ejemplo con el AWS CloudFormation script incluido.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Ejemplos de Amazon Rekognition que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante Amazon AWS SDK for .NET Rekognition.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Escenarios

Acciones

`CompareFaces`

En el siguiente ejemplo de código, se muestra cómo usar `CompareFaces`.

Para obtener información, consulte Comparación de rostros en imágenes.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to compare faces in two images.
    /// </summary>
    public class CompareFaces
    {
        public static async Task Main()
        {
            float similarityThreshold = 70F;
            string sourceImage = "source.jpg";
            string targetImage = "target.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load source image: {sourceImage}");
                return;
```

```
            }

            Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Failed to load target image: {targetImage}");
                Console.WriteLine(ex.Message);
                return;
            }

            var compareFacesRequest = new CompareFacesRequest
            {
                SourceImage = imageSource,
                TargetImage = imageTarget,
                SimilarityThreshold = similarityThreshold,
            };

            // Call operation
            var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

            // Display results
            compareFacesResponse.FaceMatches.ForEach(match =>
            {
                ComparedFace face = match.Face;
                BoundingBox position = face.BoundingBox;
                Console.WriteLine($"Face at {position.Left} {position.Top} matches
with {match.Similarity}% confidence.");
            });

            Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
face(s) that did not match.");
        }
```

```
    }
```

- Para API obtener más información, consulte CompareFaceslа AWS SDK for .NET APIReferencia.

`CreateCollection`

En el siguiente ejemplo de código, se muestra cómo usar `CreateCollection`.

Para obtener información, consulte Creación de una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Uses Amazon Rekognition to create a collection to which you can add
/// faces using the IndexFaces operation.
/// </summary>
public class CreateCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine("Creating collection: " + collectionId);

        var createCollectionRequest = new CreateCollectionRequest
        {
```

```
                CollectionId = collectionId,
            };

            CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
            Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
            Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
        }
    }
```

- Para API obtener más información, consulte CreateCollectionla AWS SDK for .NET APIReferencia.

`DeleteCollection`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteCollection`.

Para obtener información, consulte Eliminación de una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete an existing collection.
    /// </summary>
    public class DeleteCollection
    {
```

```
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Deleting collection: " + collectionId);

            var deleteCollectionRequest = new DeleteCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var deleteCollectionResponse = await
rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
            Console.WriteLine($"{collectionId}:
{deleteCollectionResponse.StatusCode}");
        }
    }
```

- Para API obtener más información, consulte DeleteCollectionla AWS SDK for .NET APIReferencia.

`DeleteFaces`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteFaces`.

Para obtener información, consulte Eliminación de rostros de una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete one or more faces from
    /// a Rekognition collection.
    /// </summary>
    public class DeleteFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" };

            var rekognitionClient = new AmazonRekognitionClient();

            var deleteFacesRequest = new DeleteFacesRequest()
            {
                CollectionId = collectionId,
                FaceIds = faces,
            };

            DeleteFacesResponse deleteFacesResponse = await
rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
            deleteFacesResponse.DeletedFaces.ForEach(face =>
            {
                Console.WriteLine($"FaceID: {face}");
            });
        }
    }
```

- Para API obtener más información, consulte DeleteFaceslа AWS SDK for .NET APIReferencia.

`DescribeCollection`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeCollection`.

Para obtener información, consulte Descripción de una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
/// collection.
/// </summary>
public class DescribeCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine($"Describing collection: {collectionId}");

        var describeCollectionRequest = new DescribeCollectionRequest()
        {
            CollectionId = collectionId,
        };

        var describeCollectionResponse = await
rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
        Console.WriteLine($"Collection ARN:
{describeCollectionResponse.CollectionARN}");
        Console.WriteLine($"Face count:
{describeCollectionResponse.FaceCount}");
        Console.WriteLine($"Face model version:
{describeCollectionResponse.FaceModelVersion}");
        Console.WriteLine($"Created:
{describeCollectionResponse.CreationTimestamp}");
    }
```

```
    }
```

- Para API obtener más información, consulte DescribeCollectionla AWS SDK for .NET APIReferencia.

`DetectFaces`

En el siguiente ejemplo de código, se muestra cómo usar `DetectFaces`.

Para obtener información, consulte Detección de rostros en una imagen.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectFaces
{
    public static async Task Main()
    {
        string photo = "input.jpg";
        string bucket = "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectFacesRequest = new DetectFacesRequest()
        {
```

```
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },

                // Attributes can be "ALL" or "DEFAULT".
                // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
                // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
items/Rekognition/TFaceDetail.html
                Attributes = new List<string>() { "ALL" },
            };

            try
            {
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
 roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
 {face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
 {face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
```

```
        }
    }
```

Visualizar información del cuadro delimitador de todos los rostros en una imagen.

```
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition\target.jpg"; //
"photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }
```

```
            int height;
            int width;

            // Used to extract original photo width/height
            using (var imageBitmap = new Bitmap(photo))
            {
                height = imageBitmap.Height;
                width = imageBitmap.Width;
            }

            Console.WriteLine("Image Information:");
            Console.WriteLine(photo);
            Console.WriteLine("Image Height: " + height);
            Console.WriteLine("Image Width: " + width);

            try
            {
                var detectFacesRequest = new DetectFacesRequest()
                {
                    Image = image,
                    Attributes = new List<string>() { "ALL" },
                };

                DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
                    Console.WriteLine($"The detected face is estimated to be between
{face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
                });
            }
            catch (Exception ex)
            {
```

```
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the image.</
param>
        /// <param name="rotation">The rotation of the face's bounding box.</param>
        public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
 BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No estimated orientation. Check Exif data.");
                return;
            }

            // Calculate face position based on image orientation.
            switch (rotation)
            {
                case "ROTATE_0":
                    left = imageWidth * box.Left;
                    top = imageHeight * box.Top;
                    break;
                case "ROTATE_90":
                    left = imageHeight * (1 - (box.Top + box.Height));
                    top = imageWidth * box.Left;
                    break;
                case "ROTATE_180":
                    left = imageWidth - (imageWidth * (box.Left + box.Width));
                    top = imageHeight * (1 - (box.Top + box.Height));
                    break;
                case "ROTATE_270":
                    left = imageHeight * box.Top;
                    top = imageWidth * (1 - box.Left - box.Width);
                    break;
                default:
```

```
                    Console.WriteLine("No estimated orientation information. Check
Exif data.");
                    return;
                }

                // Display face location information.
                Console.WriteLine($"Left: {left}");
                Console.WriteLine($"Top: {top}");
                Console.WriteLine($"Face Width: {imageWidth * box.Width}");
                Console.WriteLine($"Face Height: {imageHeight * box.Height}");
            }
        }
```

- Para API obtener más información, consulte DetectFaceslа AWS SDK for .NET APIReferencia.

`DetectLabels`

En el siguiente ejemplo de código, se muestra cómo usar `DetectLabels`.

Para obtener información, consulte Detección de etiquetas en una imagen.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class DetectLabels
```

```
    {
        public static async Task Main()
        {
            string photo = "del_river_02092020_01.jpg"; // "input.jpg";
            string bucket = "igsmiths3photos"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MaxLabels = 10,
                MinConfidence = 75F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

Detecte las etiquetas en un archivo de imagen que está almacenado en el ordenador.

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = image,
                MaxLabels = 10,
                MinConfidence = 77F,
            };

            try
            {
```

```
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- Para API obtener más información, consulte DetectLabelsla AWS SDK for .NET APIReferencia.

`DetectModerationLabels`

En el siguiente ejemplo de código, se muestra cómo usar `DetectModerationLabels`.

Para obtener información, consulte Detección de imágenes inapropiadas.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect unsafe content in a
    /// JPEG or PNG format image.
    /// </summary>
```

```
    public class DetectModerationLabels
    {
        public static async Task Main(string[] args)
        {
            string photo = "input.jpg";
            string bucket = "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectModerationLabelsRequest = new DetectModerationLabelsRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MinConfidence = 60F,
            };

            try
            {
                var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
                {
                    Console.WriteLine($"Label: {label.Name}");
                    Console.WriteLine($"Confidence: {label.Confidence}");
                    Console.WriteLine($"Parent: {label.ParentName}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- Para API obtener más información, consulte DetectModerationLabelsla AWS SDK for .NET APIReferencia.

`DetectText`

En el siguiente ejemplo de código, se muestra cómo usar `DetectText`.

Para obtener información, consulte Detección de texto en una imagen.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Uses the Amazon Rekognition Service to detect text in an image. The
/// example was created using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class DetectText
{
    public static async Task Main()
    {
        string photo = "Dad_photographer.jpg"; // "input.jpg";
        string bucket = "igsmiths3photos"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
```

```
                    Name = photo,
                    Bucket = bucket,
                },
            },
        };

        try
        {
            DetectTextResponse detectTextResponse = await
rekognitionClient.DetectTextAsync(detectTextRequest);
            Console.WriteLine($"Detected lines and words for {photo}");
            detectTextResponse.TextDetections.ForEach(text =>
            {
                Console.WriteLine($"Detected: {text.DetectedText}");
                Console.WriteLine($"Confidence: {text.Confidence}");
                Console.WriteLine($"Id : {text.Id}");
                Console.WriteLine($"Parent Id: {text.ParentId}");
                Console.WriteLine($"Type: {text.Type}");
            });
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

- Para API obtener más información, consulte DetectTextla AWS SDK for .NET APIReferencia.

`GetCelebrityInfo`

En el siguiente ejemplo de código, se muestra cómo usar `GetCelebrityInfo`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Shows how to use Amazon Rekognition to retrieve information about the
    /// celebrity identified by the supplied celebrity Id.
    /// </summary>
    public class CelebrityInfo
    {
        public static async Task Main()
        {
            string celebId = "nnnnnnnn";

            var rekognitionClient = new AmazonRekognitionClient();

            var celebrityInfoRequest = new GetCelebrityInfoRequest
            {
                Id = celebId,
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
                Console.WriteLine(url);
            });
        }
    }
```

- Para API obtener más información, consulte GetCelebrityInfola AWS SDK for .NET APIReferencia.

`IndexFaces`

En el siguiente ejemplo de código, se muestra cómo usar `IndexFaces`.

Para obtener información, consulte Adición de rostros a una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "doc-example-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };
```

```
            var indexFacesRequest = new IndexFacesRequest
            {
                Image = image,
                CollectionId = collectionId,
                ExternalImageId = photo,
                DetectionAttributes = new List<string>() { "ALL" },
            };

            IndexFacesResponse indexFacesResponse = await
rekognitionClient.IndexFacesAsync(indexFacesRequest);

            Console.WriteLine($"{photo} added");
            foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            {
                Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
            }
        }
    }
```

- Para API obtener más información, consulte IndexFaceslа AWS SDK for .NET APIReferencia.

`ListCollections`

En el siguiente ejemplo de código, se muestra cómo usar `ListCollections`.

Para obtener información, consulte Enumerar colecciones.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
```

```
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses Amazon Rekognition to list the collection IDs in the
    /// current account.
    /// </summary>
    public class ListCollections
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            Console.WriteLine("Listing collections");
            int limit = 10;

            var listCollectionsRequest = new ListCollectionsRequest
            {
                MaxResults = limit,
            };

            var listCollectionsResponse = new ListCollectionsResponse();

            do
            {
                if (listCollectionsResponse is not null)
                {
                    listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
                }

                listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

                listCollectionsResponse.CollectionIds.ForEach(id =>
                {
                    Console.WriteLine(id);
                });
            }
            while (listCollectionsResponse.NextToken is not null);
        }
    }
```

- Para API obtener más información, consulte ListCollectionsla AWS SDK for .NET APIReferencia.

`ListFaces`

En el siguiente ejemplo de código, se muestra cómo usar `ListFaces`.

Para obtener información, consulte Enumerar rostros en una colección.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Uses the Amazon Rekognition Service to retrieve the list of faces
/// stored in a collection.
/// </summary>
public class ListFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";

        var rekognitionClient = new AmazonRekognitionClient();

        var listFacesResponse = new ListFacesResponse();
        Console.WriteLine($"Faces in collection {collectionId}");

        var listFacesRequest = new ListFacesRequest
        {
            CollectionId = collectionId,
            MaxResults = 1,
        };
```

```
            do
            {
                listFacesResponse = await
rekognitionClient.ListFacesAsync(listFacesRequest);
                listFacesResponse.Faces.ForEach(face =>
                {
                    Console.WriteLine(face.FaceId);
                });

                listFacesRequest.NextToken = listFacesResponse.NextToken;
            }
            while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
        }
    }
```

- Para API obtener más información, consulte ListFacesla AWS SDK for .NET APIReferencia.

`RecognizeCelebrities`

En el siguiente ejemplo de código, se muestra cómo usar `RecognizeCelebrities`.

Para obtener información, consulte Reconocimiento de famosos en una imagen.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to identify celebrities in a photo.
```

```
   /// </summary>
   public class CelebritiesInImage
   {
       public static async Task Main(string[] args)
       {
           string photo = "moviestars.jpg";

           var rekognitionClient = new AmazonRekognitionClient();

           var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

           var img = new Amazon.Rekognition.Model.Image();
           byte[] data = null;
           try
           {
               using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
               data = new byte[fs.Length];
               fs.Read(data, 0, (int)fs.Length);
           }
           catch (Exception)
           {
               Console.WriteLine($"Failed to load file {photo}");
               return;
           }

           img.Bytes = new MemoryStream(data);
           recognizeCelebritiesRequest.Image = img;

           Console.WriteLine($"Looking for celebrities in image {photo}\n");

           var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

           Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
           recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
           {
               Console.WriteLine($"Celebrity recognized: {celeb.Name}");
               Console.WriteLine($"Celebrity ID: {celeb.Id}");
               BoundingBox boundingBox = celeb.Face.BoundingBox;
               Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
               Console.WriteLine("Further information (if available):");
```

```
                celeb.Urls.ForEach(url =>
                {
                    Console.WriteLine(url);
                });
            });


Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
        }
    }
```

- Para API obtener más información, consulte RecognizeCelebritiesla AWS SDK for .NET APIReferencia.

`SearchFaces`

En el siguiente ejemplo de código, se muestra cómo usar `SearchFaces`.

Para obtener información, consulte Búsqueda de un rostro (ID de rostro).

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to find faces in an image that
    /// match the face Id provided in the method request.
    /// </summary>
    public class SearchFacesMatchingId
```

```
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

            var rekognitionClient = new AmazonRekognitionClient();

            // Search collection for faces matching the face id.
            var searchFacesRequest = new SearchFacesRequest
            {
                CollectionId = collectionId,
                FaceId = faceId,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesResponse searchFacesResponse = await
rekognitionClient.SearchFacesAsync(searchFacesRequest);

            Console.WriteLine("Face matching faceId " + faceId);

            Console.WriteLine("Matche(s): ");
            searchFacesResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
{face.Similarity}");
            });
        }
    }
```

- Para API obtener más información, consulte SearchFaceslа AWS SDK for .NET APIReferencia.

`SearchFacesByImage`

En el siguiente ejemplo de código, se muestra cómo usar `SearchFacesByImage`.

Para obtener información, consulte Búsqueda de un rostro (imagen).

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Uses the Amazon Rekognition Service to search for images matching those
/// in a collection.
/// </summary>
public class SearchFacesMatchingImage
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string bucket = "bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        var image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var searchFacesByImageRequest = new SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
            Image = image,
            FaceMatchThreshold = 70F,
```

```
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " + photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
        }
    }
```

- Para API obtener más información, consulte SearchFacesByImagela AWS SDK for .NET APIReferencia.

Escenarios

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- APIPuerta de enlace
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Detectar objetos en imágenes

El siguiente ejemplo de código muestra cómo crear una aplicación que utilice Amazon Rekognition para detectar objetos por categoría en las imágenes.

AWS SDK for .NET

Muestra cómo usar Amazon Rekognition. NETAPIpara crear una aplicación que utilice Amazon Rekognition para identificar objetos por categoría en imágenes ubicadas en un bucket de Amazon Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (AmazonSES).

Para obtener el código fuente completo y las instrucciones sobre cómo configurarla y ejecutarla, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Ejemplos de registro de dominios de Route 53 utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes mediante el AWS SDK for .NET registro de dominios de Route 53.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Registro de dominio de Introducción a Route 53

En los siguientes ejemplos de código se muestra cómo empezar a utilizar el registro de dominio de Route 53.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
public static class HelloRoute53Domains
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon Route 53 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRoute53Domains>()
            ).Build();

        // Now the client is available for injection.
        var route53Client =
 host.Services.GetRequiredService<IAmazonRoute53Domains>();

        // You can use await and any of the async methods to get a response.
        var response = await route53Client.ListPricesAsync(new ListPricesRequest
 { Tld = "com" });
        Console.WriteLine($"Hello Amazon Route 53 Domains! Following are prices
 for .com domain operations:");
        var comPrices = response.Prices.FirstOrDefault();
        if (comPrices != null)
```

```
        {
            Console.WriteLine($"\tRegistration: {comPrices.RegistrationPrice?.Price}
 {comPrices.RegistrationPrice?.Currency}");
            Console.WriteLine($"\tRenewal: {comPrices.RenewalPrice?.Price}
 {comPrices.RenewalPrice?.Currency}");
        }
    }
}
```

- Para API obtener más información, consulte ListPricesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Enumere los dominios actuales y las operaciones del año pasado.
- Consulte la facturación del año pasado y los precios de los tipos de dominio.
- Obtención de sugerencias de dominios.
- Compruebe la disponibilidad y la transferibilidad del dominio.
- Si lo desea, solicite el registro de un dominio.
- Obtención de información de una operación.
- Si lo desea, obtenga información del dominio.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 static class Route53DomainScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
        1. List current domains.
        2. List operations in the past year.
        3. View billing for the account in the past year.
        4. View prices for domain types.
        5. Get domain suggestions.
        6. Check domain availability.
        7. Check domain transferability.
        8. Optionally, request a domain registration.
        9. Get an operation detail.
       10. Optionally, get a domain detail.
   */

    private static Route53Wrapper _route53Wrapper = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
                    .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonRoute53Domains>()
                .AddTransient<Route53Wrapper>()
            )
            .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
```

```
                true) // Optionally, load local settings.
            .Build();

        var logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger(typeof(Route53DomainScenario));

        _route53Wrapper = host.Services.GetRequiredService<Route53Wrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon Route 53 domains example
scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            await ListDomains();
            await ListOperations();
            await ListBillingRecords();
            await ListPrices();
            await ListDomainSuggestions();
            await CheckDomainAvailability();
            await CheckDomainTransferability();
            var operationId = await RequestDomainRegistration();
            await GetOperationalDetail(operationId);
            await GetDomainDetails();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The Amazon Route 53 domains example scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List account registered domains.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDomains()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List account domains.");
        var domains = await _route53Wrapper.ListDomains();
        for (int i = 0; i < domains.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {domains[i].DomainName}");
        }

        if (!domains.Any())
        {
            Console.WriteLine("\tNo domains found in this account.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain operations in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListOperations()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List account domain operations in the past year.");
        var operations = await _route53Wrapper.ListOperations(
            DateTime.Today.AddYears(-1));
        for (int i = 0; i < operations.Count; i++)
        {
            Console.WriteLine($"\tOperation Id: {operations[i].OperationId}");
            Console.WriteLine($"\tStatus: {operations[i].Status}");
            Console.WriteLine($"\tDate: {operations[i].SubmittedDate}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List billing in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListBillingRecords()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. View billing for the account in the past year.");
```

```
        var billingRecords = await _route53Wrapper.ViewBilling(
            DateTime.Today.AddYears(-1),
            DateTime.Today);
        for (int i = 0; i < billingRecords.Count; i++)
        {
            Console.WriteLine($"\tBill Date:
{billingRecords[i].BillDate.ToShortDateString()}");
            Console.WriteLine($"\tOperation: {billingRecords[i].Operation}");
            Console.WriteLine($"\tPrice: {billingRecords[i].Price}");
        }
        if (!billingRecords.Any())
        {
            Console.WriteLine("\tNo billing records found in this account for the
past year.");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List prices for a few domain types.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListPrices()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. View prices for domain types.");
        var domainTypes = new List<string> { "net", "com", "org", "co" };

        var prices = await _route53Wrapper.ListPrices(domainTypes);
        foreach (var pr in prices)
        {
            Console.WriteLine($"\tName: {pr.Name}");
            Console.WriteLine($"\tRegistration: {pr.RegistrationPrice?.Price}
{pr.RegistrationPrice?.Currency}");
            Console.WriteLine($"\tRenewal: {pr.RenewalPrice?.Price}
{pr.RenewalPrice?.Currency}");
            Console.WriteLine($"\tTransfer: {pr.TransferPrice?.Price}
{pr.TransferPrice?.Currency}");
            Console.WriteLine($"\tChange Ownership: {pr.ChangeOwnershipPrice?.Price}
{pr.ChangeOwnershipPrice?.Currency}");
            Console.WriteLine($"\tRestoration: {pr.RestorationPrice?.Price}
{pr.RestorationPrice?.Currency}");
            Console.WriteLine();
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain suggestions for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDomainSuggestions()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Get domain suggestions.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to get available domain
suggestions.");
            domainName = Console.ReadLine();
        }

        var suggestions = await _route53Wrapper.GetDomainSuggestions(domainName,
true, 5);
        foreach (var suggestion in suggestions)
        {
            Console.WriteLine($"\tSuggestion Name: {suggestion.DomainName}");
            Console.WriteLine($"\tAvailability: {suggestion.Availability}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check availability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainAvailability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Check domain availability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain availability.");
            domainName = Console.ReadLine();
        }
```

```
        var availability = await
_route53Wrapper.CheckDomainAvailability(domainName);
        Console.WriteLine($"\tAvailability: {availability}");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainTransferability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Check domain transferability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain
transferability.");
            domainName = Console.ReadLine();
        }

        var transferability = await
_route53Wrapper.CheckDomainTransferability(domainName);
        Console.WriteLine($"\tTransferability: {transferability}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> RequestDomainRegistration()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Optionally, request a domain registration.");

        Console.WriteLine($"\tNote: This example uses domain request settings in
settings.json.");
        Console.WriteLine($"\tTo change the domain registration settings, set the
values in that file.");
        Console.WriteLine($"\tRemember, registering an actual domain will incur an
account billing cost.");
```

```
        Console.WriteLine($"\tWould you like to begin a domain registration? (y/
n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase))
        {
            string domainName = _configuration["DomainName"];
            ContactDetail contact = new ContactDetail();
            contact.CountryCode =
 CountryCode.FindValue(_configuration["Contact:CountryCode"]);
            contact.ContactType =
 ContactType.FindValue(_configuration["Contact:ContactType"]);

            _configuration.GetSection("Contact").Bind(contact);

            var operationId = await _route53Wrapper.RegisterDomain(
                domainName,
                Convert.ToBoolean(_configuration["AutoRenew"]),
                Convert.ToInt32(_configuration["DurationInYears"]),
                contact);
            if (operationId != null)
            {
                Console.WriteLine(
                    $"\tRegistration requested. Operation Id: {operationId}");
            }

            return operationId;
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }

    /// <summary>
    /// Get details for an operation.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetOperationalDetail(string? operationId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Get an operation detail.");

        var operationDetails =
            await _route53Wrapper.GetOperationDetail(operationId);
```

```
        Console.WriteLine(operationDetails);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Optionally, get details for a registered domain.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> GetDomainDetails()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Get details on a domain.");

        Console.WriteLine($"\tNote: you must have a registered domain to get
 details.");
        Console.WriteLine($"\tWould you like to get domain details? (y/n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase))
        {
            string? domainName = null;
            while (domainName == null)
            {
                Console.WriteLine($"\tEnter a domain name to get details.");
                domainName = Console.ReadLine();
            }

            var domainDetails = await _route53Wrapper.GetDomainDetail(domainName);
            Console.WriteLine(domainDetails);
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }
}
```

Métodos envolventes que se utilizan en situaciones para acciones de registro de dominio de Route 53.

```
public class Route53Wrapper
{
    private readonly IAmazonRoute53Domains _amazonRoute53Domains;
    private readonly ILogger<Route53Wrapper> _logger;
    public Route53Wrapper(IAmazonRoute53Domains amazonRoute53Domains,
 ILogger<Route53Wrapper> logger)
    {
        _amazonRoute53Domains = amazonRoute53Domains;
        _logger = logger;
    }


    /// <summary>
    /// List prices for domain type operations.
    /// </summary>
    /// <param name="domainTypes">Domain types to include in the results.</param>
    /// <returns>The list of domain prices.</returns>
    public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
    {
        var results = new List<DomainPrice>();
        var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
 ListPricesRequest());
        // Get the entire list using the paginator.
        await foreach (var prices in paginatePrices.Prices)
        {
            results.Add(prices);
        }
        return results.Where(p => domainTypes.Contains(p.Name)).ToList();
    }


    /// <summary>
    /// Check the availability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for availability.</param>
    /// <returns>An availability result string.</returns>
    public async Task<string> CheckDomainAvailability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
            new CheckDomainAvailabilityRequest
            {
                DomainName = domain
            }
        );
```

```
        return result.Availability.Value;
    }


    /// <summary>
    /// Check the transferability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for transferability.</param>
    /// <returns>A transferability result string.</returns>
    public async Task<string> CheckDomainTransferability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
            new CheckDomainTransferabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Transferability.Transferable.Value;
    }


    /// <summary>
    /// Get a list of suggestions for a given domain.
    /// </summary>
    /// <param name="domain">The domain to check for suggestions.</param>
    /// <param name="onlyAvailable">If true, only returns available domains.</param>
    /// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
    /// <returns>A collection of domain suggestions.</returns>
    public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
    {
        var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
            new GetDomainSuggestionsRequest
            {
                DomainName = domain,
                OnlyAvailable = onlyAvailable,
                SuggestionCount = suggestionCount
            }
        );
        return result.SuggestionsList;
    }
```

```
    /// <summary>
    /// Get details for a domain action operation.
    /// </summary>
    /// <param name="operationId">The operational Id.</param>
    /// <returns>A string describing the operational details.</returns>
    public async Task<string> GetOperationDetail(string? operationId)
    {
        if (operationId == null)
            return "Unable to get operational details because ID is null.";
        try
        {
            var operationDetails =
                await _amazonRoute53Domains.GetOperationDetailAsync(
                    new GetOperationDetailRequest
                    {
                        OperationId = operationId
                    }
                );

            var details = $"\tOperation {operationId}:\n" +
                          $"\tFor domain {operationDetails.DomainName} on
 {operationDetails.SubmittedDate.ToShortDateString()}.\n" +
                          $"\tMessage is {operationDetails.Message}.\n" +
                          $"\tStatus is {operationDetails.Status}.\n";

            return details;
        }
        catch (AmazonRoute53DomainsException ex)
        {
            return $"Unable to get operation details. Here's why: {ex.Message}.";
        }
    }


    /// <summary>
    /// Initiate a domain registration request.
    /// </summary>
    /// <param name="contact">Contact details.</param>
    /// <param name="domainName">The domain name to register.</param>
    /// <param name="autoRenew">True if the domain should automatically renew.</
param>
    /// <param name="duration">The duration in years for the domain registration.</
param>
    /// <returns>The operation Id.</returns>
```

```
   public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
duration, ContactDetail contact)
   {
       // This example uses the same contact information for admin, registrant, and
tech contacts.
       try
       {
           var result = await _amazonRoute53Domains.RegisterDomainAsync(
               new RegisterDomainRequest()
               {
                   AdminContact = contact,
                   RegistrantContact = contact,
                   TechContact = contact,
                   DomainName = domainName,
                   AutoRenew = autoRenew,
                   DurationInYears = duration,
                   PrivacyProtectAdminContact = false,
                   PrivacyProtectRegistrantContact = false,
                   PrivacyProtectTechContact = false
               }
           );
           return result.OperationId;
       }
       catch (InvalidInputException)
       {
           _logger.LogInformation($"Unable to request registration for domain
{domainName}");
           return null;
       }
   }


   /// <summary>
   /// View billing records for the account between a start and end date.
   /// </summary>
   /// <param name="startDate">The start date for billing results.</param>
   /// <param name="endDate">The end date for billing results.</param>
   /// <returns>A collection of billing records.</returns>
   public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
   {
       var results = new List<BillingRecord>();
       var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
           new ViewBillingRequest()
```

```
            {
                Start = startDate,
                End = endDate
            });

        // Get the entire list using the paginator.
        await foreach (var billingRecords in paginateBilling.BillingRecords)
        {
            results.Add(billingRecords);
        }
        return results;
    }


    /// <summary>
    /// List the domains for the account.
    /// </summary>
    /// <returns>A collection of domain summary records.</returns>
    public async Task<List<DomainSummary>> ListDomains()
    {
        var results = new List<DomainSummary>();
        var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
            new ListDomainsRequest());

        // Get the entire list using the paginator.
        await foreach (var domain in paginateDomains.Domains)
        {
            results.Add(domain);
        }
        return results;
    }


    /// <summary>
    /// List operations for the account that are submitted after a specified date.
    /// </summary>
    /// <returns>A collection of operation summary records.</returns>
    public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
    {
        var results = new List<OperationSummary>();
        var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
            new ListOperationsRequest()
            {
```

```
                SubmittedSince = submittedSince
            });

        // Get the entire list using the paginator.
        await foreach (var operations in paginateOperations.Operations)
        {
            results.Add(operations);
        }
        return results;
    }


    /// <summary>
    /// Get details for a domain.
    /// </summary>
    /// <returns>A string with detail information about the domain.</returns>
    public async Task<string> GetDomainDetail(string domainName)
    {
        try
        {
            var result = await _amazonRoute53Domains.GetDomainDetailAsync(
                new GetDomainDetailRequest()
                {
                    DomainName = domainName
                });
            var details = $"\tDomain {domainName}:\n" +
                          $"\tCreated on {result.CreationDate.ToShortDateString()}.
\n" +
                          $"\tAdmin contact is {result.AdminContact.Email}.\n" +
                          $"\tAuto-renew is {result.AutoRenew}.\n";

            return details;
        }
        catch (InvalidInputException)
        {
            return $"Domain {domainName} was not found in your account.";
        }
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.

- CheckDomainAvailability
- CheckDomainTransferability
- GetDomainDetail
- GetDomainSuggestions
- GetOperationDetail
- ListDomains
- ListOperations
- ListPrices
- RegisterDomain
- ViewBilling

Acciones

`CheckDomainAvailability`

En el siguiente ejemplo de código, se muestra cómo usar `CheckDomainAvailability`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Check the availability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for availability.</param>
/// <returns>An availability result string.</returns>
public async Task<string> CheckDomainAvailability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
        new CheckDomainAvailabilityRequest
        {
            DomainName = domain
```

```
            }
        );
        return result.Availability.Value;
    }
```

- Para API obtener más información, consulte CheckDomainAvailabilityla AWS SDK for .NET APIReferencia.

`CheckDomainTransferability`

En el siguiente ejemplo de código, se muestra cómo usar `CheckDomainTransferability`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Check the transferability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for transferability.</param>
    /// <returns>A transferability result string.</returns>
    public async Task<string> CheckDomainTransferability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
            new CheckDomainTransferabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Transferability.Transferable.Value;
    }
```

- Para API obtener más información, consulte CheckDomainTransferabilityla AWS SDK for .NET APIReferencia.

`GetDomainDetail`

En el siguiente ejemplo de código, se muestra cómo usar `GetDomainDetail`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get details for a domain.
    /// </summary>
    /// <returns>A string with detail information about the domain.</returns>
    public async Task<string> GetDomainDetail(string domainName)
    {
        try
        {
            var result = await _amazonRoute53Domains.GetDomainDetailAsync(
                new GetDomainDetailRequest()
                {
                    DomainName = domainName
                });
            var details = $"\tDomain {domainName}:\n" +
                          $"\tCreated on {result.CreationDate.ToShortDateString()}.
\n" +
                          $"\tAdmin contact is {result.AdminContact.Email}.\n" +
                          $"\tAuto-renew is {result.AutoRenew}.\n";

            return details;
        }
        catch (InvalidInputException)
        {
            return $"Domain {domainName} was not found in your account.";
        }
    }
```

- Para API obtener más información, consulte GetDomainDetailla AWS SDK for .NET APIReferencia.

`GetDomainSuggestions`

En el siguiente ejemplo de código, se muestra cómo usar `GetDomainSuggestions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 suggestions for a given domain.
   /// </summary>
   /// <param name="domain">The domain to check for suggestions.</param>
   /// <param name="onlyAvailable">If true, only returns available domains.</param>
   /// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
   /// <returns>A collection of domain suggestions.</returns>
   public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
   {
       var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
           new GetDomainSuggestionsRequest
           {
               DomainName = domain,
               OnlyAvailable = onlyAvailable,
               SuggestionCount = suggestionCount
           }
       );
       return result.SuggestionsList;
   }
```

- Para API obtener más información, consulte GetDomainSuggestionsla AWS SDK for .NET APIReferencia.

`GetOperationDetail`

En el siguiente ejemplo de código, se muestra cómo usar `GetOperationDetail`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get details for a domain action operation.
    /// </summary>
    /// <param name="operationId">The operational Id.</param>
    /// <returns>A string describing the operational details.</returns>
    public async Task<string> GetOperationDetail(string? operationId)
    {
        if (operationId == null)
            return "Unable to get operational details because ID is null.";
        try
        {
            var operationDetails =
                await _amazonRoute53Domains.GetOperationDetailAsync(
                    new GetOperationDetailRequest
                    {
                        OperationId = operationId
                    }
                );

            var details = $"\tOperation {operationId}:\n" +
                          $"\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}.\n" +
                          $"\tMessage is {operationDetails.Message}.\n" +
                          $"\tStatus is {operationDetails.Status}.\n";

            return details;
        }
        catch (AmazonRoute53DomainsException ex)
        {
            return $"Unable to get operation details. Here's why: {ex.Message}.";
```

```
        }
    }
```

- Para API obtener más información, consulte GetOperationDetailla AWS SDK for .NET APIReferencia.

`ListDomains`

En el siguiente ejemplo de código, se muestra cómo usar `ListDomains`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// List the domains for the account.
/// </summary>
/// <returns>A collection of domain summary records.</returns>
public async Task<List<DomainSummary>> ListDomains()
{
    var results = new List<DomainSummary>();
    var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
        new ListDomainsRequest());

    // Get the entire list using the paginator.
    await foreach (var domain in paginateDomains.Domains)
    {
        results.Add(domain);
    }
    return results;
}
```

- Para API obtener más información, consulte ListDomainsla AWS SDK for .NET APIReferencia.

`ListOperations`

En el siguiente ejemplo de código, se muestra cómo usar `ListOperations`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List operations for the account that are submitted after a specified date.
    /// </summary>
    /// <returns>A collection of operation summary records.</returns>
    public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
    {
        var results = new List<OperationSummary>();
        var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
            new ListOperationsRequest()
            {
                SubmittedSince = submittedSince
            });

        // Get the entire list using the paginator.
        await foreach (var operations in paginateOperations.Operations)
        {
            results.Add(operations);
        }
        return results;
    }
```

- Para API obtener más información, consulte ListOperationsla AWS SDK for .NET APIReferencia.

`ListPrices`

En el siguiente ejemplo de código, se muestra cómo usar `ListPrices`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List prices for domain type operations.
    /// </summary>
    /// <param name="domainTypes">Domain types to include in the results.</param>
    /// <returns>The list of domain prices.</returns>
    public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
    {
        var results = new List<DomainPrice>();
        var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
ListPricesRequest());
        // Get the entire list using the paginator.
        await foreach (var prices in paginatePrices.Prices)
        {
            results.Add(prices);
        }
        return results.Where(p => domainTypes.Contains(p.Name)).ToList();
    }
```

- Para API obtener más información, consulte ListPricesla AWS SDK for .NET APIReferencia.

`RegisterDomain`

En el siguiente ejemplo de código, se muestra cómo usar `RegisterDomain`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Initiate a domain registration request.
    /// </summary>
    /// <param name="contact">Contact details.</param>
    /// <param name="domainName">The domain name to register.</param>
    /// <param name="autoRenew">True if the domain should automatically renew.</
param>
    /// <param name="duration">The duration in years for the domain registration.</
param>
    /// <returns>The operation Id.</returns>
    public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
 duration, ContactDetail contact)
    {
        // This example uses the same contact information for admin, registrant, and
 tech contacts.
        try
        {
            var result = await _amazonRoute53Domains.RegisterDomainAsync(
                new RegisterDomainRequest()
                {
                    AdminContact = contact,
                    RegistrantContact = contact,
                    TechContact = contact,
                    DomainName = domainName,
                    AutoRenew = autoRenew,
                    DurationInYears = duration,
                    PrivacyProtectAdminContact = false,
                    PrivacyProtectRegistrantContact = false,
                    PrivacyProtectTechContact = false
                }
            );
            return result.OperationId;
        }
```

```
        catch (InvalidInputException)
        {
            _logger.LogInformation($"Unable to request registration for domain
{domainName}");
            return null;
        }
    }
```

- Para API obtener más información, consulte RegisterDomainla AWS SDK for .NET APIReferencia.

`ViewBilling`

En el siguiente ejemplo de código, se muestra cómo usar `ViewBilling`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// View billing records for the account between a start and end date.
    /// </summary>
    /// <param name="startDate">The start date for billing results.</param>
    /// <param name="endDate">The end date for billing results.</param>
    /// <returns>A collection of billing records.</returns>
    public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
    {
        var results = new List<BillingRecord>();
        var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
            new ViewBillingRequest()
            {
                Start = startDate,
                End = endDate
            });
```

```
        // Get the entire list using the paginator.
        await foreach (var billingRecords in paginateBilling.BillingRecords)
        {
            results.Add(billingRecords);
        }
        return results;
    }
```

- Para API obtener más información, consulte ViewBillingla AWS SDK for .NET APIReferencia.

Ejemplos de Amazon S3 que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar situaciones comunes AWS SDK for .NET mediante 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 funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Conceptos básicos
- Acciones
- Escenarios
- Ejemplos sin servidor

Conceptos básicos

Aprenda los 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.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
public class S3_Basics
{
    public static async Task Main()
    {
        // Create an Amazon S3 client object. The constructor uses the
        // default user installed on the system. To work with Amazon S3
        // features in a different AWS Region, pass the AWS Region as a
        // parameter to the client constructor.
        IAmazonS3 client = new AmazonS3Client();
        string bucketName = string.Empty;
        string filePath = string.Empty;
        string keyName = string.Empty;

        var sepBar = new string('-', Console.WindowWidth);

        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);

            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            Console.Write("Please enter a name for the new bucket: ");
            bucketName = Console.ReadLine();

            var success = await S3Bucket.CreateBucketAsync(client, 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);

            // 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;
                }
            }
```

```
            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await S3Bucket.UploadFileAsync(client, bucketName, keyName,
filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set the file path to an empty string to avoid overwriting the
            // file we just uploaded to the bucket.
            filePath = string.Empty;

            // Now get a new location where we can save the file.
            while (string.IsNullOrEmpty(filePath))
            {
                // First get the path to which the file will be downloaded.
                Console.Write("Please enter the path where the file will be
downloaded: ");
                filePath = Console.ReadLine();

                // Confirm that the file exists on the local computer.
                if (File.Exists($"{filePath}\\{keyName}"))
                {
                    Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                    filePath = string.Empty;
                }
            }

            // Download an object from a bucket.
            success = await S3Bucket.DownloadObjectFromBucketAsync(client,
bucketName, keyName, filePath);

            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;

            while (string.IsNullOrEmpty(folderName))
            {
                Console.Write("Please enter the name of the folder to copy your
object to: ");
                folderName = Console.ReadLine();
            }

            while (string.IsNullOrEmpty(keyName))
            {
                // Get the name to give to the object once uploaded.
                Console.Write("Enter the name of the object to copy: ");
                keyName = Console.ReadLine();
            }

            await S3Bucket.CopyObjectInBucketAsync(client, bucketName, keyName,
folderName);

            // List the objects in the bucket.
            await S3Bucket.ListBucketContentsAsync(client, bucketName);

            // Delete the contents of the bucket.
            await S3Bucket.DeleteBucketContentsAsync(client, bucketName);

            // Deleting the bucket too quickly after deleting its contents will
            // cause an error that the bucket isn't empty. So...
            Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
            _ = Console.ReadLine();

            // Delete the bucket.
            await S3Bucket.DeleteBucketAsync(client, bucketName);
        }
    }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Acciones

`AbortMultipartUploads`

En el siguiente ejemplo de código, se muestra cómo usar `AbortMultipartUploads`.

AWS SDK for .NET

> **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 System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to use the Amazon Simple Storage Service
/// (Amazon S3) to stop a multi-part upload process using the Amazon S3
/// TransferUtility.
/// </summary>
public class AbortMPU
{
```

```
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";

            // If the AWS Region defined for your default user is different
            // from the Region where your Amazon S3 bucket is located,
            // pass the Region name to the S3 client object's constructor.
            // For example: RegionEndpoint.USWest2.
            IAmazonS3 client = new AmazonS3Client();

            await AbortMPUAsync(client, bucketName);
        }

        /// <summary>
        /// Cancels the multi-part copy process.
        /// </summary>
        /// <param name="client">The initialized client object used to create
        /// the TransferUtility object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the
        /// multi-part copy operation is in progress.</param>
        public static async Task AbortMPUAsync(IAmazonS3 client, string bucketName)
        {
            try
            {
                var transferUtility = new TransferUtility(client);

                // Cancel all in-progress uploads initiated before the specified
date.
                await transferUtility.AbortMultipartUploadsAsync(
                    bucketName, DateTime.Now.AddDays(-7));
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine($"Error: {e.Message}");
            }
        }
    }
```

- Para API obtener más información, consulte AbortMultipartUploadsla AWS SDK for .NET APIReferencia.

`CopyObject`

En el siguiente ejemplo de código, se muestra cómo usar `CopyObject`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    public class CopyObject
    {
        public static async Task Main()
        {
            // Specify the AWS Region where your buckets are located if it is
            // different from the AWS Region of the default user.
            IAmazonS3 s3Client = new AmazonS3Client();

            // Remember to change these values to refer to your Amazon S3 objects.
            string sourceBucketName = "doc-example-bucket1";
            string destinationBucketName = "doc-example-bucket2";
            string sourceObjectKey = "testfile.txt";
            string destinationObjectKey = "testfilecopy.txt";

            Console.WriteLine($"Copying {sourceObjectKey} from {sourceBucketName} to
");
            Console.WriteLine($"{destinationBucketName} as {destinationObjectKey}");

            var response = await CopyingObjectAsync(
                s3Client,
                sourceObjectKey,
                destinationObjectKey,
                sourceBucketName,
                destinationBucketName);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
            {
                Console.WriteLine("\nCopy complete.");
            }
        }

        /// <summary>
        /// This method calls the AWS SDK for .NET to copy an
        /// object from one Amazon S3 bucket to another.
        /// </summary>
        /// <param name="client">The Amazon S3 client object.</param>
        /// <param name="sourceKey">The name of the object to be copied.</param>
        /// <param name="destinationKey">The name under which to save the copy.</
param>
        /// <param name="sourceBucketName">The name of the Amazon S3 bucket
        /// where the file is located now.</param>
        /// <param name="destinationBucketName">The name of the Amazon S3
        /// bucket where the copy should be saved.</param>
        /// <returns>Returns a CopyObjectResponse object with the results from
        /// the async call.</returns>
        public static async Task<CopyObjectResponse> CopyingObjectAsync(
            IAmazonS3 client,
            string sourceKey,
            string destinationKey,
            string sourceBucketName,
            string destinationBucketName)
        {
            var response = new CopyObjectResponse();
            try
            {
                var request = new CopyObjectRequest
                {
                    SourceBucket = sourceBucketName,
                    SourceKey = sourceKey,
                    DestinationBucket = destinationBucketName,
                    DestinationKey = destinationKey,
                };
                response = await client.CopyObjectAsync(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error copying object: '{ex.Message}'");
            }

            return response;
```

```
        }
    }
```

- Para API obtener más información, consulte CopyObjectla AWS SDK for .NET APIReferencia.

`CreateBucket`

En el siguiente ejemplo de código, se muestra cómo usar `CreateBucket`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> CreateBucketAsync(IAmazonS3 client, string
bucketName)
        {
            try
            {
                var request = new PutBucketRequest
                {
                    BucketName = bucketName,
                    UseClientRegion = true,
                };

                var response = await client.PutBucketAsync(request);
                return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
            }
```

```
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error creating bucket: '{ex.Message}'");
                return false;
            }
        }
```

Cree un nuevo bucket con el bloqueo de objetos habilitado.

```
    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
 enableObjectLock)
    {
        Console.WriteLine($"\tCreating bucket {bucketName} with object lock
 {enableObjectLock}.");
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
                ObjectLockEnabledForBucket = enableObjectLock,
            };

            var response = await _amazonS3.PutBucketAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- Para API obtener más información, consulte CreateBucketla AWS SDK for .NET APIReferencia.

`DeleteBucket`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteBucket`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DeleteBucketAsync(IAmazonS3 client, string
 bucketName)
        {
            var request = new DeleteBucketRequest
            {
                BucketName = bucketName,
            };

            var response = await client.DeleteBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para API obtener más información, consulte DeleteBucketla AWS SDK for .NET APIReferencia.

`DeleteBucketCors`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteBucketCors`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Deletes a CORS configuration from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to delete the CORS configuration from the bucket.</param>
        private static async Task DeleteCORSConfigurationAsync(AmazonS3Client
client)
        {
            DeleteCORSConfigurationRequest request = new
DeleteCORSConfigurationRequest()
            {
                BucketName = BucketName,
            };
            await client.DeleteCORSConfigurationAsync(request);
        }
```

- Para API obtener más información, consulte DeleteBucketCorsla AWS SDK for .NET APIReferencia.

`DeleteBucketLifecycle`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteBucketLifecycle`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// This method removes the Lifecycle configuration from the named
        /// S3 bucket.
        /// </summary>
        /// <param name="client">The S3 client object used to call
        /// the RemoveLifecycleConfigAsync method.</param>
        /// <param name="bucketName">A string representing the name of the
        /// S3 bucket from which the configuration will be removed.</param>
        public static async Task RemoveLifecycleConfigAsync(IAmazonS3 client, string
bucketName)
        {
            var request = new DeleteLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
            };
            await client.DeleteLifecycleConfigurationAsync(request);
        }
```

- Para API obtener más información, consulte DeleteBucketLifecyclela AWS SDK for .NET APIReferencia.

`DeleteObject`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteObject`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Elimine un objeto de un bucket de S3 no versionado.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete an object from a non-versioned Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteObject
{
    /// <summary>
    /// The Main method initializes the necessary variables and then calls
    /// the DeleteObjectNonVersionedBucketAsync method to delete the object
    /// named by the keyName parameter.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";
        const string keyName = "testfile.txt";

        // If the Amazon S3 bucket is located in an AWS Region other than the
        // Region of the default account, define the AWS Region for the
        // Amazon S3 bucket in your call to the AmazonS3Client constructor.
        // For example RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
        await DeleteObjectNonVersionedBucketAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// The DeleteObjectNonVersionedBucketAsync takes care of deleting the
    /// desired object from the named bucket.
    /// </summary>
```

```
        /// <param name="client">An initialized Amazon S3 client used to delete
        /// an object from an Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the bucket from which the
        /// object will be deleted.</param>
        /// <param name="keyName">The name of the object to delete.</param>
        public static async Task DeleteObjectNonVersionedBucketAsync(IAmazonS3
client, string bucketName, string keyName)
        {
            try
            {
                var deleteObjectRequest = new DeleteObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                };

                Console.WriteLine($"Deleting object: {keyName}");
                await client.DeleteObjectAsync(deleteObjectRequest);
                Console.WriteLine($"Object: {keyName} deleted from {bucketName}.");
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error encountered on server.
Message:'{ex.Message}' when deleting an object.");
            }
        }
    }
```

Elimine un objeto de un bucket de S3 versionado.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example creates an object in an Amazon Simple Storage Service
    /// (Amazon S3) bucket and then deletes the object version that was
    /// created.
    /// </summary>
    public class DeleteObjectVersion
```

```
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "verstioned-object.txt";

            // If the AWS Region of the default user is different from the AWS
            // Region of the Amazon S3 bucket, pass the AWS Region of the
            // bucket region to the Amazon S3 client object's constructor.
            // Define it like this:
            //      RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
            IAmazonS3 client = new AmazonS3Client();

            await CreateAndDeleteObjectVersionAsync(client, bucketName, keyName);
        }

        /// <summary>
        /// This method creates and then deletes a versioned object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
        /// create and delete the object.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
        /// object will be created and deleted.</param>
        /// <param name="keyName">The key name of the object to create.</param>
        public static async Task CreateAndDeleteObjectVersionAsync(IAmazonS3 client,
string bucketName, string keyName)
        {
            try
            {
                // Add a sample object.
                string versionID = await PutAnObject(client, bucketName, keyName);

                // Delete the object by specifying an object key and a version ID.
                DeleteObjectRequest request = new DeleteObjectRequest()
                {
                    BucketName = bucketName,
                    Key = keyName,
                    VersionId = versionID,
                };

                Console.WriteLine("Deleting an object");
                await client.DeleteObjectAsync(request);
            }
            catch (AmazonS3Exception ex)
```

```
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }

        /// <summary>
        /// This method is used to create the temporary Amazon S3 object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 object which will be used
        /// to create the temporary Amazon S3 object.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
 object
        /// will be created.</param>
        /// <param name="objectKey">The name of the Amazon S3 object co create.</
param>
        /// <returns>The Version ID of the created object.</returns>
        public static async Task<string> PutAnObject(IAmazonS3 client, string
 bucketName, string objectKey)
        {
            PutObjectRequest request = new PutObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                ContentBody = "This is the content body!",
            };

            PutObjectResponse response = await client.PutObjectAsync(request);
            return response.VersionId;
        }
    }
```

- Para API obtener más información, consulte DeleteObjectla AWS SDK for .NET APIReferencia.

`DeleteObjects`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteObjects`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Elimine todos los objetos de un bucket de S3.

```
        /// <summary>
        /// Delete all of the objects stored in an existing Amazon S3 bucket.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DeleteBucketContentsAsync(IAmazonS3 client,
string bucketName)
        {
            // Iterate over the contents of the bucket and delete all objects.
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
            };

            try
            {
                ListObjectsV2Response response;

                do
                {
                    response = await client.ListObjectsV2Async(request);
                    response.S3Objects
                        .ForEach(async obj => await
client.DeleteObjectAsync(bucketName, obj.Key));

                    // If the response is truncated, set the request
ContinuationToken
                    // from the NextContinuationToken property of the response.
                    request.ContinuationToken = response.NextContinuationToken;
```

```
            }
            while (response.IsTruncated);

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

Elimine varios objetos de un bucket de S3 no versionado.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete multiple objects from an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    /// <summary>
    /// The Main method initializes the Amazon S3 client and the name of
    /// the bucket and then passes those values to MultiObjectDeleteAsync.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";

        // If the Amazon S3 bucket from which you wish to delete objects is not
        // located in the same AWS Region as the default user, define the
        // AWS Region for the Amazon S3 bucket as a parameter to the client
        // constructor.
        IAmazonS3 s3Client = new AmazonS3Client();

        await MultiObjectDeleteAsync(s3Client, bucketName);
```

```
        }

        /// <summary>
        /// This method uses the passed Amazon S3 client to first create and then
        /// delete three files from the named bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// Amazon S3 methods.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where objects
        /// will be created and then deleted.</param>
        public static async Task MultiObjectDeleteAsync(IAmazonS3 client, string
bucketName)
        {
            // Create three sample objects which we will then delete.
            var keysAndVersions = await PutObjectsAsync(client, 3, bucketName);

            // Now perform the multi-object delete, passing the key names and
            // version IDs. Since we are working with a non-versioned bucket,
            // the object keys collection includes null version IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keysAndVersions,
            };

            // You can add a specific object key to the delete request using the
            // AddKey method of the multiObjectDeleteRequest.
            try
            {
                DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionErrorStatus(e);
            }
        }

        /// <summary>
        /// Prints the list of errors raised by the call to DeleteObjectsAsync.
        /// </summary>
```

```
        /// <param name="ex">A collection of exceptions returned by the call to
        /// DeleteObjectsAsync.</param>
        public static void PrintDeletionErrorStatus(DeleteObjectsException ex)
        {
            DeleteObjectsResponse errorResponse = ex.Response;
            Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

            Console.WriteLine($"Successfully deleted
 {errorResponse.DeletedObjects.Count}.");
            Console.WriteLine($"No. of objects failed to delete =
 {errorResponse.DeleteErrors.Count}");

            Console.WriteLine("Printing error data...");
            foreach (DeleteError deleteError in errorResponse.DeleteErrors)
            {
                Console.WriteLine($"Object Key:
 {deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
            }
        }

        /// <summary>
        /// This method creates simple text file objects that can be used in
        /// the delete method.
        /// </summary>
        /// <param name="client">The Amazon S3 client used to call PutObjectAsync.</
param>
        /// <param name="number">The number of objects to create.</param>
        /// <param name="bucketName">The name of the bucket where the objects
        /// will be created.</param>
        /// <returns>A list of keys (object keys) and versions that the calling
        /// method will use to delete the newly created files.</returns>
        public static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3 client,
 int number, string bucketName)
        {
            List<KeyVersion> keys = new List<KeyVersion>();
            for (int i = 0; i < number; i++)
            {
                string key = "ExampleObject-" + new System.Random().Next();
                PutObjectRequest request = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = key,
                    ContentBody = "This is the content body!",
                };
```

```
            PutObjectResponse response = await client.PutObjectAsync(request);

            // For non-versioned bucket operations, we only need the
            // object key.
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
            };
            keys.Add(keyVersion);
        }

        return keys;
    }
}
```

Elimine varios objetos de un bucket de S3 versionado.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete objects in a version-enabled Amazon
/// Simple StorageService (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region for your Amazon S3 bucket is different from
        // the AWS Region of the default user, define the AWS Region for
        // the Amazon S3 bucket and pass it to the client constructor
        // like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        IAmazonS3 s3Client;
```

```
            s3Client = new AmazonS3Client();
            await DeleteMultipleObjectsFromVersionedBucketAsync(s3Client,
bucketName);
        }

        /// <summary>
        /// This method removes multiple versions and objects from a
        /// version-enabled Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        public static async Task
DeleteMultipleObjectsFromVersionedBucketAsync(IAmazonS3 client, string bucketName)
        {
            // Delete objects (specifying object version in the request).
            await DeleteObjectVersionsAsync(client, bucketName);

            // Delete objects (without specifying object version in the request).
            var deletedObjects = await DeleteObjectsAsync(client, bucketName);

            // Additional exercise - remove the delete markers Amazon S3 returned
from
            // the preceding response. This results in the objects reappearing
            // in the bucket (you can verify the appearance/disappearance of
            // objects in the console).
            await RemoveDeleteMarkersAsync(client, bucketName, deletedObjects);
        }

        /// <summary>
        /// Creates and then deletes non-versioned Amazon S3 objects and then
deletes
        /// them again. The method returns a list of the Amazon S3 objects deleted.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// PubObjectsAsync and NonVersionedDeleteAsync.</param>
        /// <param name="bucketName">The name of the bucket where the objects
        /// will be created and then deleted.</param>
        /// <returns>A list of DeletedObjects.</returns>
```

```
        public static async Task<List<DeletedObject>> DeleteObjectsAsync(IAmazonS3
client, string bucketName)
        {
            // Upload the sample objects.
            var keysAndVersions2 = await PutObjectsAsync(client, bucketName, 3);

            // Delete objects using only keys. Amazon S3 creates a delete marker and
            // returns its version ID in the response.
            List<DeletedObject> deletedObjects = await
NonVersionedDeleteAsync(client, bucketName, keysAndVersions2);
            return deletedObjects;
        }

        /// <summary>
        /// This method creates several temporary objects and then deletes them.
        /// </summary>
        /// <param name="client">The S3 client.</param>
        /// <param name="bucketName">Name of the bucket.</param>
        /// <returns>Async task.</returns>
        public static async Task DeleteObjectVersionsAsync(IAmazonS3 client, string
bucketName)
        {
            // Upload the sample objects.
            var keysAndVersions1 = await PutObjectsAsync(client, bucketName, 3);

            // Delete the specific object versions.
            await VersionedDeleteAsync(client, bucketName, keysAndVersions1);
        }

        /// <summary>
        /// Displays the list of information about deleted files to the console.
        /// </summary>
        /// <param name="e">Error information from the delete process.</param>
        private static void DisplayDeletionErrors(DeleteObjectsException e)
        {
            var errorResponse = e.Response;
            Console.WriteLine($"No. of objects successfully deleted =
{errorResponse.DeletedObjects.Count}");
            Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");
            Console.WriteLine("Printing error data...");
            foreach (var deleteError in errorResponse.DeleteErrors)
            {
```

```
                Console.WriteLine($"Object Key:
 {deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
            }
        }

        /// <summary>
        /// Delete multiple objects from a version-enabled bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="keys">A list of key names for the objects to delete.</
param>
        private static async Task VersionedDeleteAsync(IAmazonS3 client, string
 bucketName, List<KeyVersion> keys)
        {
            var multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keys, // This includes the object keys and specific
 version IDs.
            };

            try
            {
                Console.WriteLine("Executing VersionedDelete...");
                DeleteObjectsResponse response = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine($"Successfully deleted all the
 {response.DeletedObjects.Count} items");
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
            }
        }

        /// <summary>
        /// Deletes multiple objects from a non-versioned Amazon S3 bucket.
        /// </summary>
```

```
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="keys">A list of key names for the objects to delete.</
param>
        /// <returns>A list of the deleted objects.</returns>
        private static async Task<List<DeletedObject>>
 NonVersionedDeleteAsync(IAmazonS3 client, string bucketName, List<KeyVersion> keys)
        {
            // Create a request that includes only the object key names.
            DeleteObjectsRequest multiObjectDeleteRequest = new
 DeleteObjectsRequest();
            multiObjectDeleteRequest.BucketName = bucketName;

            foreach (var key in keys)
            {
                multiObjectDeleteRequest.AddKey(key.Key);
            }

            // Execute DeleteObjectsAsync.
            // The DeleteObjectsAsync method adds a delete marker for each
            // object deleted. You can verify that the objects were removed
            // using the Amazon S3 console.
            DeleteObjectsResponse response;
            try
            {
                Console.WriteLine("Executing NonVersionedDelete...");
                response = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
 response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
                throw; // Some deletions failed. Investigate before continuing.
            }

            // This response contains the DeletedObjects list which we use to delete
 the delete markers.
            return response.DeletedObjects;
```

```
        }

        /// <summary>
        /// Deletes the markers left after deleting the temporary objects.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="deletedObjects">A list of the objects that were deleted.</
param>
        private static async Task RemoveDeleteMarkersAsync(IAmazonS3 client, string
 bucketName, List<DeletedObject> deletedObjects)
        {
            var keyVersionList = new List<KeyVersion>();

            foreach (var deletedObject in deletedObjects)
            {
                KeyVersion keyVersion = new KeyVersion
                {
                    Key = deletedObject.Key,
                    VersionId = deletedObject.DeleteMarkerVersionId,
                };
                keyVersionList.Add(keyVersion);
            }

            // Create another request to delete the delete markers.
            var multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keyVersionList,
            };

            // Now, delete the delete marker to bring your objects back to the
 bucket.
            try
            {
                Console.WriteLine("Removing the delete markers .....");
                var deleteObjectResponse = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine($"Successfully deleted the
 {deleteObjectResponse.DeletedObjects.Count} delete markers");
```

```
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
            }
        }

        /// <summary>
        /// Create temporary Amazon S3 objects to show how object deletion wors in
an
        /// Amazon S3 bucket with versioning enabled.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// PutObjectAsync to create temporary objects for the example.</param>
        /// <param name="bucketName">A string representing the name of the S3
        /// bucket where we will create the temporary objects.</param>
        /// <param name="number">The number of temporary objects to create.</param>
        /// <returns>A list of the KeyVersion objects.</returns>
        private static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3
client, string bucketName, int number)
        {
            var keys = new List<KeyVersion>();

            for (var i = 0; i < number; i++)
            {
                string key = "ObjectToDelete-" + new System.Random().Next();
                PutObjectRequest request = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = key,
                    ContentBody = "This is the content body!",
                };

                var response = await client.PutObjectAsync(request);
                KeyVersion keyVersion = new KeyVersion
                {
                    Key = key,
                    VersionId = response.VersionId,
                };

                keys.Add(keyVersion);
            }
```

```
            return keys;
        }
    }
```

- Para API obtener más información, consulte DeleteObjectsla AWS SDK for .NET APIReferencia.

`GetBucketAcl`

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketAcl`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Get the access control list (ACL) for the new bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to get the
        /// access control list (ACL) of the bucket.</param>
        /// <param name="newBucketName">The name of the newly created bucket.</
param>
        /// <returns>An S3AccessControlList.</returns>
        public static async Task<S3AccessControlList> GetACLForBucketAsync(IAmazonS3
 client, string newBucketName)
        {
            // Retrieve bucket ACL to show that the ACL was properly applied to
            // the new bucket.
            GetACLResponse getACLResponse = await client.GetACLAsync(new
 GetACLRequest
            {
                BucketName = newBucketName,
            });
```

```
            return getACLResponse.AccessControlList;
        }
```

- Para API obtener más información, consulte GetBucketAcll la AWS SDK for .NET APIReferencia.

`GetBucketCors`

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketCors`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Retrieve the CORS configuration applied to the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to retrieve the CORS configuration.</param>
        /// <returns>The created CORS configuration object.</returns>
        private static async Task<CORSConfiguration>
RetrieveCORSConfigurationAsync(AmazonS3Client client)
        {
            GetCORSConfigurationRequest request = new GetCORSConfigurationRequest()
            {
                BucketName = BucketName,
            };
            var response = await client.GetCORSConfigurationAsync(request);
            var configuration = response.Configuration;
            PrintCORSRules(configuration);
            return configuration;
        }
```

- Para API obtener más información, consulte GetBucketCorsla AWS SDK for .NET APIReferencia.

`GetBucketLifecycleConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketLifecycleConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Returns a configuration object for the supplied bucket name.
        /// </summary>
        /// <param name="client">The S3 client object used to call
        /// the GetLifecycleConfigurationAsync method.</param>
        /// <param name="bucketName">The name of the S3 bucket for which a
        /// configuration will be created.</param>
        /// <returns>Returns a new LifecycleConfiguration object.</returns>
        public static async Task<LifecycleConfiguration>
RetrieveLifecycleConfigAsync(IAmazonS3 client, string bucketName)
        {
            var request = new GetLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
            };
            var response = await client.GetLifecycleConfigurationAsync(request);
            var configuration = response.Configuration;
            return configuration;
        }
```

- Para API obtener más información, consulte GetBucketLifecycleConfigurationla AWS SDK for .NET APIReferencia.

`GetBucketWebsite`

En el siguiente ejemplo de código, se muestra cómo usar `GetBucketWebsite`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
                // Get the website configuration.
                GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
                {
                    BucketName = bucketName,
                };
                GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
                Console.WriteLine($"Index document:
{getResponse.WebsiteConfiguration.IndexDocumentSuffix}");
                Console.WriteLine($"Error document:
{getResponse.WebsiteConfiguration.ErrorDocument}");
```

- Para API obtener más información, consulte GetBucketWebsitela AWS SDK for .NET APIReferencia.

`GetObject`

En el siguiente ejemplo de código, se muestra cómo usar `GetObject`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DownloadObjectFromBucketAsync(
            IAmazonS3 client,
            string bucketName,
            string objectName,
            string filePath)
        {
            // Create a GetObject request
            var request = new GetObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
            };

            // Issue request and remember to dispose of the response
            using GetObjectResponse response = await client.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 API obtener más información, consulte GetObjectla AWS SDK for .NET APIReferencia.

`GetObjectLegalHold`

En el siguiente ejemplo de código, se muestra cómo usar `GetObjectLegalHold`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
```

```
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }
```

- Para API obtener más información, consulte GetObjectLegalHoldla AWS SDK for .NET APIReferencia.

`GetObjectLockConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `GetObjectLockConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>
    /// <returns>The bucket's object lock configuration details.</returns>
    public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
    {
        try
        {
            var request = new GetObjectLockConfigurationRequest()
            {
                BucketName = bucketName
            };

            var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tBucket object lock config for {bucketName} in
{bucketName}: " +
                              $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
```

```
                                $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

            return response.ObjectLockConfiguration;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock config:
'{ex.Message}'");
            return new ObjectLockConfiguration();
        }
    }
```

- Para API obtener más información, consulte GetObjectLockConfigurationla AWS SDK for .NET APIReferencia.

`GetObjectRetention`

En el siguiente ejemplo de código, se muestra cómo usar `GetObjectRetention`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the retention period for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object retention details.</returns>
    public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectRetentionRequest()
```

```
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectRetentionAsync(request);
            Console.WriteLine($"\tObject retention for {objectKey} in {bucketName}:
" +
                              $"\n\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
            return response.Retention;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock retention:
'{ex.Message}'");
            return new ObjectLockRetention();
        }
    }
```

- Para API obtener más información, consulte GetObjectRetentionla AWS SDK for .NET APIReferencia.

`ListBuckets`

En el siguiente ejemplo de código, se muestra cómo usar `ListBuckets`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace ListBucketsExample
{
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
```

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

    /// <summary>
    /// This example uses the AWS SDK for .NET to list the Amazon Simple Storage
    /// Service (Amazon S3) buckets belonging to the default account.
    /// </summary>
    public class ListBuckets
    {
        private static IAmazonS3 _s3Client;

        /// <summary>
        /// Get a list of the buckets owned by the default user.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client object.</param>
        /// <returns>The response from the ListingBuckets call that contains a
        /// list of the buckets owned by the default user.</returns>
        public static async Task<ListBucketsResponse> GetBuckets(IAmazonS3 client)
        {
            return await client.ListBucketsAsync();
        }

        /// <summary>
        /// This method lists the name and creation date for the buckets in
        /// the passed List of S3 buckets.
        /// </summary>
        /// <param name="bucketList">A List of S3 bucket objects.</param>
        public static void DisplayBucketList(List<S3Bucket> bucketList)
        {
            bucketList
                .ForEach(b => Console.WriteLine($"Bucket name: {b.BucketName},
created on: {b.CreationDate}"));
        }

        public static async Task Main()
        {
            // The client uses the AWS Region of the default user.
            // If the Region where the buckets were created is different,
            // pass the Region to the client constructor. For example:
            // _s3Client = new AmazonS3Client(RegionEndpoint.USEast1);
            _s3Client = new AmazonS3Client();
            var response = await GetBuckets(_s3Client);
            DisplayBucketList(response.Buckets);
        }
```

```
    }
}
```

- Para API obtener más información, consulte ListBucketsla AWS SDK for .NET APIReferencia.

`ListObjectVersions`

En el siguiente ejemplo de código, se muestra cómo usar `ListObjectVersions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example lists the versions of the objects in a version enabled
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class ListObjectVersions
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region where your bucket is defined is different from
        // the AWS Region where the Amazon S3 bucket is defined, pass the
constant
        // for the AWS Region to the client constructor like this:
        //      var client = new AmazonS3Client(RegionEndpoint.USWest2);
        IAmazonS3 client = new AmazonS3Client();
        await GetObjectListWithAllVersionsAsync(client, bucketName);
    }
```

```
        /// <summary>
        /// This method lists all versions of the objects within an Amazon S3
        /// version enabled bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// ListVersionsAsync.</param>
        /// <param name="bucketName">The name of the version enabled Amazon S3
 bucket
        /// for which you want to list the versions of the contained objects.</
param>
        public static async Task GetObjectListWithAllVersionsAsync(IAmazonS3 client,
 string bucketName)
        {
            try
            {
                // When you instantiate the ListVersionRequest, you can
                // optionally specify a key name prefix in the request
                // if you want a list of object versions of a specific object.

                // For this example we set a small limit in MaxKeys to return
                // a small list of versions.
                ListVersionsRequest request = new ListVersionsRequest()
                {
                    BucketName = bucketName,
                    MaxKeys = 2,
                };

                do
                {
                    ListVersionsResponse response = await
 client.ListVersionsAsync(request);

                    // Process response.
                    foreach (S3ObjectVersion entry in response.Versions)
                    {
                        Console.WriteLine($"key: {entry.Key} size: {entry.Size}");
                    }

                    // If response is truncated, set the marker to get the next
                    // set of keys.
                    if (response.IsTruncated)
                    {
                        request.KeyMarker = response.NextKeyMarker;
```

```
                    request.VersionIdMarker = response.NextVersionIdMarker;
                }
                else
                {
                    request = null;
                }
            }
            while (request != null);
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error: '{ex.Message}'");
        }
    }
}
```

- Para API obtener más información, consulte ListObjectVersionsla AWS SDK for .NET APIReferencia.

`ListObjectsV2`

En el siguiente ejemplo de código, se muestra cómo usar `ListObjectsV2`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon S3 client object.</param>
    /// <param name="bucketName">The name of the bucket for which to list
    /// the contents.</param>
    /// <returns>A boolean value indicating the success or failure of the
```

```
        /// copy operation.</returns>
        public static async Task<bool> ListBucketContentsAsync(IAmazonS3 client,
 string bucketName)
        {
            try
            {
                var request = new ListObjectsV2Request
                {
                    BucketName = bucketName,
                    MaxKeys = 5,
                };

                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");

                ListObjectsV2Response response;

                do
                {
                    response = await client.ListObjectsV2Async(request);

                    response.S3Objects
                        .ForEach(obj => Console.WriteLine($"{obj.Key,-35}
{obj.LastModified.ToShortDateString(),10}{obj.Size,10}"));

                    // If the response is truncated, set the request
 ContinuationToken
                    // from the NextContinuationToken property of the response.
                    request.ContinuationToken = response.NextContinuationToken;
                }
                while (response.IsTruncated);

                return true;
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error encountered on server.
 Message:'{ex.Message}' getting list of objects.");
                return false;
            }
        }
```

Muestre objetos con un paginador.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// The following example lists objects in an Amazon Simple Storage
    /// Service (Amazon S3) bucket.
    /// </summary>
    public class ListObjectsPaginator
    {
        private const string BucketName = "doc-example-bucket";

        public static async Task Main()
        {
            IAmazonS3 s3Client = new AmazonS3Client();

            Console.WriteLine($"Listing the objects contained in {BucketName}:\n");
            await ListingObjectsAsync(s3Client, BucketName);
        }

        /// <summary>
        /// This method uses a paginator to retrieve the list of objects in an
        /// an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">An Amazon S3 client object.</param>
        /// <param name="bucketName">The name of the S3 bucket whose objects
        /// you want to list.</param>
        public static async Task ListingObjectsAsync(IAmazonS3 client, string
bucketName)
        {
            var listObjectsV2Paginator = client.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });

            await foreach (var response in listObjectsV2Paginator.Responses)
            {
```

```
                Console.WriteLine($"HttpStatusCode: {response.HttpStatusCode}");
                Console.WriteLine($"Number of Keys: {response.KeyCount}");
                foreach (var entry in response.S3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }
        }
    }
```

- Para API obtener más información, consulte ListObjectsV2 en AWS SDK for .NET APIla referencia.

`PutBucketAccelerateConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketAccelerateConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Amazon Simple Storage Service (Amazon S3) Transfer Acceleration is a
/// bucket-level feature that enables you to perform faster data transfers
/// to Amazon S3. This example shows how to configure Transfer
/// Acceleration.
/// </summary>
public class TransferAcceleration
{
```

```
        /// <summary>
        /// The main method initializes the client object and sets the
        /// Amazon Simple Storage Service (Amazon S3) bucket name before
        /// calling EnableAccelerationAsync.
        /// </summary>
        public static async Task Main()
        {
            var s3Client = new AmazonS3Client();
            const string bucketName = "doc-example-bucket";

            await EnableAccelerationAsync(s3Client, bucketName);
        }

        /// <summary>
        /// This method sets the configuration to enable transfer acceleration
        /// for the bucket referred to in the bucketName parameter.
        /// </summary>
        /// <param name="client">An Amazon S3 client used to enable the
        /// acceleration on an Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket for which the
        /// method will be enabling acceleration.</param>
        private static async Task EnableAccelerationAsync(AmazonS3Client client,
string bucketName)
        {
            try
            {
                var putRequest = new PutBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                    AccelerateConfiguration = new AccelerateConfiguration
                    {
                        Status = BucketAccelerateStatus.Enabled,
                    },
                };
                await client.PutBucketAccelerateConfigurationAsync(putRequest);

                var getRequest = new GetBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                };
                var response = await
client.GetBucketAccelerateConfigurationAsync(getRequest);

                Console.WriteLine($"Acceleration state = '{response.Status}' ");
```

```
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error occurred. Message:'{ex.Message}' when
 setting transfer acceleration");
            }
        }
    }
```

- Para API obtener más información, consulte PutBucketAccelerateConfigurationla AWS SDK for .NET APIReferencia.

`PutBucketAcl`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketAcl`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Creates an Amazon S3 bucket with an ACL to control access to the
        /// bucket and the objects stored in it.
        /// </summary>
        /// <param name="client">The initialized client object used to create
        /// an Amazon S3 bucket, with an ACL applied to the bucket.
        /// </param>
        /// <param name="region">The AWS Region where the bucket will be created.</
param>
        /// <param name="newBucketName">The name of the bucket to create.</param>
        /// <returns>A boolean value indicating success or failure.</returns>
        public static async Task<bool> CreateBucketUseCannedACLAsync(IAmazonS3
 client, S3Region region, string newBucketName)
        {
```

```
            try
            {
                // Create a new Amazon S3 bucket with Canned ACL.
                var putBucketRequest = new PutBucketRequest()
                {
                    BucketName = newBucketName,
                    BucketRegion = region,
                    CannedACL = S3CannedACL.LogDeliveryWrite,
                };

                PutBucketResponse putBucketResponse = await
client.PutBucketAsync(putBucketRequest);

                return putBucketResponse.HttpStatusCode ==
System.Net.HttpStatusCode.OK;
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Amazon S3 error: {ex.Message}");
            }

            return false;
        }
```

- Para API obtener más información, consulte PutBucketAclla AWS SDK for .NET APIReferencia.

`PutBucketCors`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketCors`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
```

```
        /// Add CORS configuration to the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to apply the CORS configuration to an Amazon S3 bucket.</param>
        /// <param name="configuration">The CORS configuration to apply.</param>
        private static async Task PutCORSConfigurationAsync(AmazonS3Client client,
CORSConfiguration configuration)
        {
            PutCORSConfigurationRequest request = new PutCORSConfigurationRequest()
            {
                BucketName = BucketName,
                Configuration = configuration,
            };

            _ = await client.PutCORSConfigurationAsync(request);
        }
```

- Para API obtener más información, consulte PutBucketCorsla AWS SDK for .NET APIReferencia.

`PutBucketLifecycleConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketLifecycleConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        /// <summary>
        /// Adds lifecycle configuration information to the S3 bucket named in
        /// the bucketName parameter.
        /// </summary>
        /// <param name="client">The S3 client used to call the
        /// PutLifecycleConfigurationAsync method.</param>
```

```
        /// <param name="bucketName">A string representing the S3 bucket to
        /// which configuration information will be added.</param>
        /// <param name="configuration">A LifecycleConfiguration object that
        /// will be applied to the S3 bucket.</param>
        public static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
string bucketName, LifecycleConfiguration configuration)
        {
            var request = new PutLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
                Configuration = configuration,
            };
            var response = await client.PutLifecycleConfigurationAsync(request);
        }
```

- Para API obtener más información, consulte PutBucketLifecycleConfigurationla AWS SDK for .NET APIReferencia.

`PutBucketLogging`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketLogging`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;
    using Microsoft.Extensions.Configuration;

    /// <summary>
    /// This example shows how to enable logging on an Amazon Simple Storage
```

```
    /// Service (Amazon S3) bucket. You need to have two Amazon S3 buckets for
    /// this example. The first is the bucket for which you wish to enable
    /// logging, and the second is the location where you want to store the
    /// logs.
    /// </summary>
    public class ServerAccessLogging
    {
        private static IConfiguration _configuration = null!;

        public static async Task Main()
        {
            LoadConfig();

            string bucketName = _configuration["BucketName"];
            string logBucketName = _configuration["LogBucketName"];
            string logObjectKeyPrefix = _configuration["LogObjectKeyPrefix"];
            string accountId = _configuration["AccountId"];

            // If the AWS Region defined for your default user is different
            // from the Region where your Amazon S3 bucket is located,
            // pass the Region name to the Amazon S3 client object's constructor.
            // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
            IAmazonS3 client = new AmazonS3Client();

            try
            {
                // Update bucket policy for target bucket to allow delivery of logs
to it.
                await SetBucketPolicyToAllowLogDelivery(
                    client,
                    bucketName,
                    logBucketName,
                    logObjectKeyPrefix,
                    accountId);

                // Enable logging on the source bucket.
                await EnableLoggingAsync(
                    client,
                    bucketName,
                    logBucketName,
                    logObjectKeyPrefix);
            }
            catch (AmazonS3Exception e)
            {
```

```
                Console.WriteLine($"Error: {e.Message}");
            }
        }

        /// <summary>
        /// This method grants appropriate permissions for logging to the
        /// Amazon S3 bucket where the logs will be stored.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client which will be used
        /// to apply the bucket policy.</param>
        /// <param name="sourceBucketName">The name of the source bucket.</param>
        /// <param name="logBucketName">The name of the bucket where logging
        /// information will be stored.</param>
        /// <param name="logPrefix">The logging prefix where the logs should be
delivered.</param>
        /// <param name="accountId">The account id of the account where the source
bucket exists.</param>
        /// <returns>Async task.</returns>
        public static async Task SetBucketPolicyToAllowLogDelivery(
            IAmazonS3 client,
            string sourceBucketName,
            string logBucketName,
            string logPrefix,
            string accountId)
        {
            var resourceArn = @"""arn:aws:s3:::" + logBucketName + "/" + logPrefix +
@"*""";

            var newPolicy = @"{
                                ""Statement"":[{
                                ""Sid"": ""S3ServerAccessLogsPolicy"",
                                ""Effect"": ""Allow"",
                                ""Principal"": { ""Service"":
""logging.s3.amazonaws.com"" },
                                ""Action"": [""s3:PutObject""],
                                ""Resource"": [" + resourceArn + @"],
                                ""Condition"": {
                                ""ArnLike"": { ""aws:SourceArn"": ""arn:aws:s3:::" +
sourceBucketName + @""" },
                                ""StringEquals"": { ""aws:SourceAccount"": """ +
accountId + @""" }
                                        }
                                    }]
                                }";
```

```
            Console.WriteLine($"The policy to apply to bucket {logBucketName} to
enable logging:");
            Console.WriteLine(newPolicy);

            PutBucketPolicyRequest putRequest = new PutBucketPolicyRequest
            {
                BucketName = logBucketName,
                Policy = newPolicy,
            };
            await client.PutBucketPolicyAsync(putRequest);
            Console.WriteLine("Policy applied.");
        }

        /// <summary>
        /// This method enables logging for an Amazon S3 bucket. Logs will be stored
        /// in the bucket you selected for logging. Selected prefix
        /// will be prepended to each log object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client which will be used
        /// to configure and apply logging to the selected Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket for which you
        /// wish to enable logging.</param>
        /// <param name="logBucketName">The name of the Amazon S3 bucket where
logging
        /// information will be stored.</param>
        /// <param name="logObjectKeyPrefix">The prefix to prepend to each
        /// object key.</param>
        /// <returns>Async task.</returns>
        public static async Task EnableLoggingAsync(
            IAmazonS3 client,
            string bucketName,
            string logBucketName,
            string logObjectKeyPrefix)
        {
            Console.WriteLine($"Enabling logging for bucket {bucketName}.");
            var loggingConfig = new S3BucketLoggingConfig
            {
                TargetBucketName = logBucketName,
                TargetPrefix = logObjectKeyPrefix,
            };

            var putBucketLoggingRequest = new PutBucketLoggingRequest
            {
                BucketName = bucketName,
```

```
                LoggingConfig = loggingConfig,
            };
            await client.PutBucketLoggingAsync(putBucketLoggingRequest);
            Console.WriteLine($"Logging enabled.");
        }

        /// <summary>
        /// Loads configuration from settings files.
        /// </summary>
        public static void LoadConfig()
        {
            _configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("settings.json") // Load settings from .json file.
                .AddJsonFile("settings.local.json", true) // Optionally, load local
settings.
                .Build();
        }
    }
```

- Para API obtener más información, consulte PutBucketLoggingla AWS SDK for .NET APIReferencia.

`PutBucketNotificationConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketNotificationConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
```

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

    /// <summary>
    /// This example shows how to enable notifications for an Amazon Simple
    /// Storage Service (Amazon S3) bucket.
    /// </summary>
    public class EnableNotifications
    {
        public static async Task Main()
        {
            const string bucketName = "doc-example-bucket1";
            const string snsTopic = "arn:aws:sns:us-east-2:0123456789ab:bucket-
notify";
            const string sqsQueue = "arn:aws:sqs:us-
east-2:0123456789ab:Example_Queue";

            IAmazonS3 client = new AmazonS3Client(Amazon.RegionEndpoint.USEast2);
            await EnableNotificationAsync(client, bucketName, snsTopic, sqsQueue);
        }

        /// <summary>
        /// This method makes the call to the PutBucketNotificationAsync method.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client used to call
        /// the PutBucketNotificationAsync method.</param>
        /// <param name="bucketName">The name of the bucket for which
        /// notifications will be turned on.</param>
        /// <param name="snsTopic">The ARN for the Amazon Simple Notification
        /// Service (Amazon SNS) topic associated with the S3 bucket.</param>
        /// <param name="sqsQueue">The ARN of the Amazon Simple Queue Service
        /// (Amazon SQS) queue to which notifications will be pushed.</param>
        public static async Task EnableNotificationAsync(
            IAmazonS3 client,
            string bucketName,
            string snsTopic,
            string sqsQueue)
        {
            try
            {
                // The bucket for which we are setting up notifications.
                var request = new PutBucketNotificationRequest()
                {
                    BucketName = bucketName,
```

```
                };

                // Defines the topic to use when sending a notification.
                var topicConfig = new TopicConfiguration()
                {
                    Events = new List<EventType> { EventType.ObjectCreatedCopy },
                    Topic = snsTopic,
                };
                request.TopicConfigurations = new List<TopicConfiguration>
                {
                    topicConfig,
                };
                request.QueueConfigurations = new List<QueueConfiguration>
                {
                    new QueueConfiguration()
                    {
                        Events = new List<EventType> { EventType.ObjectCreatedPut },
                        Queue = sqsQueue,
                    },
                };

                // Now apply the notification settings to the bucket.
                PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
```

- Para API obtener más información, consulte PutBucketNotificationConfigurationla AWS SDK for .NET APIReferencia.

`PutBucketWebsite`

En el siguiente ejemplo de código, se muestra cómo usar `PutBucketWebsite`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
                // Put the website configuration.
                PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
                {
                    BucketName = bucketName,
                    WebsiteConfiguration = new WebsiteConfiguration()
                    {
                        IndexDocumentSuffix = indexDocumentSuffix,
                        ErrorDocument = errorDocument,
                    },
                };
                PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- Para API obtener más información, consulte PutBucketWebsitela AWS SDK for .NET APIReferencia.

`PutObject`

En el siguiente ejemplo de código, se muestra cómo usar `PutObject`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> UploadFileAsync(
            IAmazonS3 client,
            string bucketName,
            string objectName,
            string filePath)
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await client.PutObjectAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully uploaded {objectName} to
{bucketName}.");
                return true;
            }
            else
            {
                Console.WriteLine($"Could not upload {objectName} to
{bucketName}.");
                return false;
            }
        }
```

Cargar un objeto con cifrado del lado del servidor.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to upload an object to an Amazon Simple Storage
    /// Service (Amazon S3) bucket with server-side encryption enabled.
    /// </summary>
    public class ServerSideEncryption
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "samplefile.txt";

            // If the AWS Region defined for your default user is different
            // from the Region where your Amazon S3 bucket is located,
            // pass the Region name to the Amazon S3 client object's constructor.
            // For example: RegionEndpoint.USWest2.
            IAmazonS3 client = new AmazonS3Client();

            await WritingAnObjectAsync(client, bucketName, keyName);
        }

        /// <summary>
        /// Upload a sample object include a setting for encryption.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
        /// to upload a file and apply server-side encryption.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
        /// encrypted object will reside.</param>
        /// <param name="keyName">The name for the object that you want to
        /// create in the supplied bucket.</param>
        public static async Task WritingAnObjectAsync(IAmazonS3 client, string
bucketName, string keyName)
        {
            try
            {
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
```

```
                    ContentBody = "sample text",
                    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256,
                };

                var putResponse = await client.PutObjectAsync(putRequest);

                // Determine the encryption state of an object.
                GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                };
                GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
                ServerSideEncryptionMethod objectEncryption =
response.ServerSideEncryptionMethod;

                Console.WriteLine($"Encryption method used: {0}",
objectEncryption.ToString());
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: '{ex.Message}' when writing an object");
            }
        }
    }
```

- Para API obtener más información, consulte PutObjectla AWS SDK for .NET APIReferencia.

`PutObjectLegalHold`

En el siguiente ejemplo de código, se muestra cómo usar `PutObjectLegalHold`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Set or modify a legal hold on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="holdStatus">The On or Off status for the legal hold.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectLegalHold(string bucketName,
        string objectKey, ObjectLockLegalHoldStatus holdStatus)
    {
        try
        {
            var request = new PutObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                LegalHold = new ObjectLockLegalHold()
                {
                    Status = holdStatus
                }
            };

            var response = await _amazonS3.PutObjectLegalHoldAsync(request);
            Console.WriteLine($"\tModified legal hold for {objectKey} in
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying legal hold: '{ex.Message}'");
            return false;
        }
    }
```

- Para API obtener más información, consulte PutObjectLegalHoldla AWS SDK for .NET APIReferencia.

`PutObjectLockConfiguration`

En el siguiente ejemplo de código, se muestra cómo usar `PutObjectLockConfiguration`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Establecimiento de la configuración de bloqueo de objetos de un bucket

```
    /// <summary>
    /// Enable object lock on an existing bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to modify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableObjectLockOnBucket(string bucketName)
    {
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
                },
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded an object lock policy to bucket
{bucketName}.");
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
            return false;
        }
    }
```

Establecimiento del período de retención predeterminado de un bucket

```
    /// <summary>
    /// Set or modify a retention period on an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to modify.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date for retention until.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
    {
        var enabledString = enableObjectLock ? "Enabled" : "Disabled";
        var timeDifference = retainUntilDate.Subtract(DateTime.Now);
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
```

```
                    ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                    Rule = new ObjectLockRule()
                    {
                        DefaultRetention = new DefaultRetention()
                        {
                            Mode = retention,
                            Days = timeDifference.Days // Can be specified in days
or years but not both.
                        }
                    }
                }
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded a default retention to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying object lock: '{ex.Message}'");
            return false;
        }
    }
```

- Para API obtener más información, consulte PutObjectLockConfigurationla AWS SDK for .NET APIReferencia.

`PutObjectRetention`

En el siguiente ejemplo de código, se muestra cómo usar `PutObjectRetention`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Set or modify a retention period on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date retention expires.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
        string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
    {
        try
        {
            var request = new PutObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Retention = new ObjectLockRetention()
                {
                    Mode = retention,
                    RetainUntilDate = retainUntilDate
                }
            };

            var response = await _amazonS3.PutObjectRetentionAsync(request);
            Console.WriteLine($"\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying retention period:
'{ex.Message}'");
            return false;
        }
    }
```

- Para API obtener más información, consulte PutObjectRetentionla AWS SDK for .NET APIReferencia.

`RestoreObject`

En el siguiente ejemplo de código, se muestra cómo usar `RestoreObject`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to restore an archived object in an Amazon
    /// Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class RestoreArchivedObject
    {
        public static void Main()
        {
            string bucketName = "doc-example-bucket";
            string objectKey = "archived-object.txt";

            // Specify your bucket region (an example region is shown).
            RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

            IAmazonS3 client = new AmazonS3Client(bucketRegion);
            RestoreObjectAsync(client, bucketName, objectKey).Wait();
        }

        /// <summary>
        /// This method restores an archived object from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// RestoreObjectAsync.</param>
        /// <param name="bucketName">A string representing the name of the
```

```
        /// bucket where the object was located before it was archived.</param>
        /// <param name="objectKey">A string representing the name of the
        /// archived object to restore.</param>
        public static async Task RestoreObjectAsync(IAmazonS3 client, string
bucketName, string objectKey)
        {
            try
            {
                var restoreRequest = new RestoreObjectRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Days = 2,
                };
                RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

                // Check the status of the restoration.
                await CheckRestorationStatusAsync(client, bucketName, objectKey);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine($"Error: {amazonS3Exception.Message}");
            }
        }

        /// <summary>
        /// This method retrieves the status of the object's restoration.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// GetObjectMetadataAsync.</param>
        /// <param name="bucketName">A string representing the name of the Amazon
        /// S3 bucket which contains the archived object.</param>
        /// <param name="objectKey">A string representing the name of the
        /// archived object you want to restore.</param>
        public static async Task CheckRestorationStatusAsync(IAmazonS3 client,
string bucketName, string objectKey)
        {
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
```

```
            };

            GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);

            var restStatus = response.RestoreInProgress ? "in-progress" : "finished
or failed";
            Console.WriteLine($"Restoration status: {restStatus}");
        }
    }
```

- Para API obtener más información, consulte RestoreObjectla AWS SDK for .NET APIReferencia.

Escenarios

Cree un prefirmado URL

El siguiente ejemplo de código muestra cómo crear un objeto prefirmado URL para Amazon S3 y cómo cargar un objeto.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Genere un prefirmado URL que pueda realizar una acción de Amazon S3 durante un tiempo limitado.

```
    using System;
    using Amazon;
    using Amazon.S3;
    using Amazon.S3.Model;

    public class GenPresignedUrl
    {
```

```
        public static void Main()
        {
            const string bucketName = "doc-example-bucket";
            const string objectKey = "sample.txt";

            // Specify how long the presigned URL lasts, in hours
            const double timeoutDuration = 12;

            // Specify the AWS Region of your Amazon S3 bucket. If it is
            // different from the Region defined for the default user,
            // pass the Region to the constructor for the client. For
            // example: new AmazonS3Client(RegionEndpoint.USEast1);

            // If using the Region us-east-1, and server-side encryption with AWS
 KMS, you must specify Signature Version 4.
            // Region us-east-1 defaults to Signature Version 2 unless explicitly
 set to Version 4 as shown below.
            // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
userguide/UsingAWSSDK.html#specify-signature-version
            // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
TAWSConfigsS3.html
            AWSConfigsS3.UseSignatureVersion4 = true;
            IAmazonS3 s3Client = new AmazonS3Client(RegionEndpoint.USEast1);

            string urlString = GeneratePresignedURL(s3Client, bucketName, objectKey,
 timeoutDuration);
            Console.WriteLine($"The generated URL is: {urlString}.");
        }

        /// <summary>
        /// Generate a presigned URL that can be used to access the file named
        /// in the objectKey parameter for the amount of time specified in the
        /// duration parameter.
        /// </summary>
        /// <param name="client">An initialized S3 client object used to call
        /// the GetPresignedUrl method.</param>
        /// <param name="bucketName">The name of the S3 bucket containing the
        /// object for which to create the presigned URL.</param>
        /// <param name="objectKey">The name of the object to access with the
        /// presigned URL.</param>
        /// <param name="duration">The length of time for which the presigned
        /// URL will be valid.</param>
        /// <returns>A string representing the generated presigned URL.</returns>
```

```
        public static string GeneratePresignedURL(IAmazonS3 client, string
bucketName, string objectKey, double duration)
        {
            string urlString = string.Empty;
            try
            {
                var request = new GetPreSignedUrlRequest()
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Expires = DateTime.UtcNow.AddHours(duration),
                };
                urlString = client.GetPreSignedURL(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error:'{ex.Message}'");
            }

            return urlString;
        }
    }
```

Genere un prefirmado URL y realice una carga con él. URL

```
    using System;
    using System.IO;
    using System.Net.Http;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to upload an object to an Amazon Simple Storage
    /// Service (Amazon S3) bucket using a presigned URL. The code first
    /// creates a presigned URL and then uses it to upload an object to an
    /// Amazon S3 bucket using that URL.
    /// </summary>
    public class UploadUsingPresignedURL
    {
```

```
        private static HttpClient httpClient = new HttpClient();

        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "samplefile.txt";
            string filePath = $"source\\{keyName}";

            // Specify how long the signed URL will be valid in hours.
            double timeoutDuration = 12;

            // Specify the AWS Region of your Amazon S3 bucket. If it is
            // different from the Region defined for the default user,
            // pass the Region to the constructor for the client. For
            // example: new AmazonS3Client(RegionEndpoint.USEast1);

            // If using the Region us-east-1, and server-side encryption with AWS
 KMS, you must specify Signature Version 4.
            // Region us-east-1 defaults to Signature Version 2 unless explicitly
 set to Version 4 as shown below.
            // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
userguide/UsingAWSSDK.html#specify-signature-version
            // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
TAWSConfigsS3.html
            AWSConfigsS3.UseSignatureVersion4 = true;
            IAmazonS3 client = new AmazonS3Client(RegionEndpoint.USEast1);

            var url = GeneratePreSignedURL(client, bucketName, keyName,
 timeoutDuration);
            var success = await UploadObject(filePath, url);

            if (success)
            {
                Console.WriteLine("Upload succeeded.");
            }
            else
            {
                Console.WriteLine("Upload failed.");
            }
        }

        /// <summary>
        /// Uploads an object to an Amazon S3 bucket using the presigned URL passed
 in
```

```
        /// the url parameter.
        /// </summary>
        /// <param name="filePath">The path (including file name) to the local
        /// file you want to upload.</param>
        /// <param name="url">The presigned URL that will be used to upload the
        /// file to the Amazon S3 bucket.</param>
        /// <returns>A Boolean value indicating the success or failure of the
        /// operation, based on the HttpWebResponse.</returns>
        public static async Task<bool> UploadObject(string filePath, string url)
        {
            using var streamContent = new StreamContent(
                new FileStream(filePath, FileMode.Open, FileAccess.Read));

            var response = await httpClient.PutAsync(url, streamContent);
            return response.IsSuccessStatusCode;
        }

        /// <summary>
        /// Generates a presigned URL which will be used to upload an object to
        /// an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// GetPreSignedURL.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket to which the
        /// presigned URL will point.</param>
        /// <param name="objectKey">The name of the file that will be uploaded.</
param>
        /// <param name="duration">How long (in hours) the presigned URL will
        /// be valid.</param>
        /// <returns>The generated URL.</returns>
        public static string GeneratePreSignedURL(
            IAmazonS3 client,
            string bucketName,
            string objectKey,
            double duration)
        {
            var request = new GetPreSignedUrlRequest
            {
                BucketName = bucketName,
                Key = objectKey,
                Verb = HttpVerb.PUT,
                Expires = DateTime.UtcNow.AddHours(duration),
            };
```

```
            string url = client.GetPreSignedURL(request);
            return url;
        }
    }
```

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- APIPuerta de enlace
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Detectar objetos en imágenes

El siguiente ejemplo de código muestra cómo crear una aplicación que utilice Amazon Rekognition para detectar objetos por categoría en las imágenes.

AWS SDK for .NET

Muestra cómo usar Amazon Rekognition. NETAPIpara crear una aplicación que utilice Amazon Rekognition para identificar objetos por categoría en imágenes ubicadas en un bucket de Amazon Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (AmazonSES).

Para obtener el código fuente completo y las instrucciones sobre cómo configurarla y ejecutarla, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Introducción al cifrado

El siguiente ejemplo de código muestra cómo empezar a cifrar objetos de Amazon S3.

AWS SDK for .NET

> **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 System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to apply client encryption to an object in an
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class SSEClientEncryption
{
```

```
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "exampleobject.txt";
            string copyTargetKeyName = "examplecopy.txt";

            // If the AWS Region defined for your default user is different
            // from the Region where your Amazon S3 bucket is located,
            // pass the Region name to the Amazon S3 client object's constructor.
            // For example: RegionEndpoint.USWest2.
            IAmazonS3 client = new AmazonS3Client();

            try
            {
                // Create an encryption key.
                Aes aesEncryption = Aes.Create();
                aesEncryption.KeySize = 256;
                aesEncryption.GenerateKey();
                string base64Key = Convert.ToBase64String(aesEncryption.Key);

                // Upload the object.
                PutObjectRequest putObjectRequest = await UploadObjectAsync(client,
bucketName, keyName, base64Key);

                // Download the object and verify that its contents match what you
uploaded.
                await DownloadObjectAsync(client, bucketName, keyName, base64Key,
putObjectRequest);

                // Get object metadata and verify that the object uses AES-256
encryption.
                await GetObjectMetadataAsync(client, bucketName, keyName,
base64Key);

                // Copy both the source and target objects using server-side
encryption with
                // an encryption key.
                await CopyObjectAsync(client, bucketName, keyName,
copyTargetKeyName, aesEncryption, base64Key);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
```

```
        }

        /// <summary>
        /// Uploads an object to an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// PutObjectAsync.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket to which the
        /// object will be uploaded.</param>
        /// <param name="keyName">The name of the object to upload to the Amazon S3
        /// bucket.</param>
        /// <param name="base64Key">The encryption key.</param>
        /// <returns>The PutObjectRequest object for use by DownloadObjectAsync.</
returns>
        public static async Task<PutObjectRequest> UploadObjectAsync(
            IAmazonS3 client,
            string bucketName,
            string keyName,
            string base64Key)
        {
            PutObjectRequest putObjectRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
                ServerSideEncryptionCustomerMethod =
 ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };
            PutObjectResponse putObjectResponse = await
 client.PutObjectAsync(putObjectRequest);
            return putObjectRequest;
        }

        /// <summary>
        /// Downloads an encrypted object from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// GetObjectAsync.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
 object
        /// is located.</param>
```

```
        /// <param name="keyName">The name of the Amazon S3 object to download.</
param>
        /// <param name="base64Key">The encryption key used to encrypt the
        /// object.</param>
        /// <param name="putObjectRequest">The PutObjectRequest used to upload
        /// the object.</param>
        public static async Task DownloadObjectAsync(
            IAmazonS3 client,
            string bucketName,
            string keyName,
            string base64Key,
            PutObjectRequest putObjectRequest)
        {
            GetObjectRequest getObjectRequest = new GetObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,

                // Provide encryption information for the object stored in Amazon
 S3.
                ServerSideEncryptionCustomerMethod =
 ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            using (GetObjectResponse getResponse = await
 client.GetObjectAsync(getObjectRequest))
            using (StreamReader reader = new
 StreamReader(getResponse.ResponseStream))
            {
                string content = reader.ReadToEnd();
                if (string.Compare(putObjectRequest.ContentBody, content) == 0)
                {
                    Console.WriteLine("Object content is same as we uploaded");
                }
                else
                {
                    Console.WriteLine("Error...Object content is not same.");
                }

                if (getResponse.ServerSideEncryptionCustomerMethod ==
 ServerSideEncryptionCustomerMethod.AES256)
                {
```

```
                Console.WriteLine("Object encryption method is AES256, same as
we set");
            }
            else
            {
                Console.WriteLine("Error...Object encryption method is not the
same as AES256 we set");
            }
        }
    }

    /// <summary>
    /// Retrieves the metadata associated with an Amazon S3 object.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used
    /// to call GetObjectMetadataAsync.</param>
    /// <param name="bucketName">The name of the Amazon S3 bucket containing the
    /// object for which we want to retrieve metadata.</param>
    /// <param name="keyName">The name of the object for which we wish to
    /// retrieve the metadata.</param>
    /// <param name="base64Key">The encryption key associated with the
    /// object.</param>
    public static async Task GetObjectMetadataAsync(
        IAmazonS3 client,
        string bucketName,
        string keyName,
        string base64Key)
    {
        GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
        {
            BucketName = bucketName,
            Key = keyName,

            // The object stored in Amazon S3 is encrypted, so provide the
necessary encryption information.
            ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key,
        };

        GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
```

```
            Console.WriteLine("The object metadata show encryption method used is:
{0}", getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
        }

        /// <summary>
        /// Copies an encrypted object from one Amazon S3 bucket to another.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// CopyObjectAsync.</param>
        /// <param name="bucketName">The Amazon S3 bucket containing the object
        /// to copy.</param>
        /// <param name="keyName">The name of the object to copy.</param>
        /// <param name="copyTargetKeyName">The Amazon S3 bucket to which the object
        /// will be copied.</param>
        /// <param name="aesEncryption">The encryption type to use.</param>
        /// <param name="base64Key">The encryption key to use.</param>
        public static async Task CopyObjectAsync(
            IAmazonS3 client,
            string bucketName,
            string keyName,
            string copyTargetKeyName,
            Aes aesEncryption,
            string base64Key)
        {
            aesEncryption.GenerateKey();
            string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

            CopyObjectRequest copyRequest = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = keyName,
                DestinationBucket = bucketName,
                DestinationKey = copyTargetKeyName,

                // Information about the source object's encryption.
                CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,

                // Information about the target object's encryption.
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = copyBase64Key,
```

```
            };
            await client.CopyObjectAsync(copyRequest);
        }
    }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CopyObject
  - GetObject
  - GetObjectMetadata

Comenzar a utilizar etiquetas

El ejemplo de código siguiente muestra cómo empezar a usar etiquetas para objetos de Amazon S3.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to work with tags in Amazon Simple Storage
/// Service (Amazon S3) objects.
/// </summary>
public class ObjectTag
{
    public static async Task Main()
    {
```

```
            string bucketName = "doc-example-bucket";
            string keyName = "newobject.txt";
            string filePath = @"*** file path ***";

            // Specify your bucket region (an example region is shown).
            RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

            var client = new AmazonS3Client(bucketRegion);
            await PutObjectsWithTagsAsync(client, bucketName, keyName, filePath);
        }

        /// <summary>
        /// This method uploads an object with tags. It then shows the tag
        /// values, changes the tags, and shows the new tags.
        /// </summary>
        /// <param name="client">The Initialized Amazon S3 client object used
        /// to call the methods to create and change an objects tags.</param>
        /// <param name="bucketName">A string representing the name of the
        /// bucket where the object will be stored.</param>
        /// <param name="keyName">A string representing the key name of the
        /// object to be tagged.</param>
        /// <param name="filePath">The directory location and file name of the
        /// object to be uploaded to the Amazon S3 bucket.</param>
        public static async Task PutObjectsWithTagsAsync(IAmazonS3 client, string
bucketName, string keyName, string filePath)
        {
            try
            {
                // Create an object with tags.
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    FilePath = filePath,
                    TagSet = new List<Tag>
                    {
                        new Tag { Key = "Keyx1", Value = "Value1" },
                        new Tag { Key = "Keyx2", Value = "Value2" },
                    },
                };

                PutObjectResponse response = await
client.PutObjectAsync(putRequest);
```

```
            // Now retrieve the new object's tags.
            GetObjectTaggingRequest getTagsRequest = new
GetObjectTaggingRequest()
            {
                BucketName = bucketName,
                Key = keyName,
            };

            GetObjectTaggingResponse objectTags = await
client.GetObjectTaggingAsync(getTagsRequest);

            // Display the tag values.
            objectTags.Tagging
                .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));

            Tagging newTagSet = new Tagging()
            {
                TagSet = new List<Tag>
                {
                    new Tag { Key = "Key3", Value = "Value3" },
                    new Tag { Key = "Key4", Value = "Value4" },
                },
            };

            PutObjectTaggingRequest putObjTagsRequest = new
PutObjectTaggingRequest()
            {
                BucketName = bucketName,
                Key = keyName,
                Tagging = newTagSet,
            };

            PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

            // Retrieve the tags again and show the values.
            GetObjectTaggingRequest getTagsRequest2 = new
GetObjectTaggingRequest()
            {
                BucketName = bucketName,
                Key = keyName,
            };
```

```
                GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);

                objectTags2.Tagging
                    .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine(
                        $"Error: '{ex.Message}'");
            }
        }
    }
```

- Para API obtener más información, consulte GetObjectTaggingla AWS SDK for .NET APIReferencia.

Obtención de la configuración de retención legal de un objeto

En el siguiente ejemplo de código, se muestra cómo obtener la configuración de retención legal de un bucket de S3.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
```

```
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }
```

- Para API obtener más información, consulte GetObjectLegalHoldla AWS SDK for .NET APIReferencia.

Bloqueo de objetos de Amazon S3

En el siguiente ejemplo de código, se muestra cómo trabajar con características de bloqueo de objetos de S3.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecute un escenario interactivo en el que se demuestren las características de bloqueo de objetos de Amazon S3.

```
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace S3ObjectLockScenario;

public static class S3ObjectLockWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
        1. Create test Amazon Simple Storage Service (S3) buckets with different
 lock policies.
        2. Upload sample objects to each bucket.
        3. Set some Legal Hold and Retention Periods on objects and buckets.
        4. Investigate lock policies by viewing settings or attempting to delete or
 overwrite objects.
        5. Clean up objects and buckets.
   */

    public static S3ActionsWrapper _s3ActionsWrapper = null!;
    public static IConfiguration _configuration = null!;
    private static string _resourcePrefix = null!;
    private static string noLockBucketName = null!;
    private static string lockEnabledBucketName = null!;
    private static string retentionAfterCreationBucketName = null!;
    private static List<string> bucketNames = new List<string>();
    private static List<string> fileNames = new List<string>();

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
```

```
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3ActionsWrapper>()
            )
            .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        ConfigurationSetup();

        ServicesSetup(host);

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Amazon Simple Storage Service (S3)
Object Locking Workflow Scenario.");
            Console.WriteLine(new string('-', 80));
            await Setup(true);

            await DemoActionChoices();

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Cleaning up resources.");
            Console.WriteLine(new string('-', 80));
            await Cleanup(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon S3 Object Locking Workflow is complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem: {ex.Message}");
            await Cleanup(true);
```

```
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _s3ActionsWrapper = host.Services.GetRequiredService<S3ActionsWrapper>();
    }

    /// <summary>
    /// Any setup operations needed.
    /// </summary>
    public static void ConfigurationSetup()
    {
        _resourcePrefix = _configuration["resourcePrefix"] ?? "dotnet-example";

        noLockBucketName = _resourcePrefix + "-no-lock";
        lockEnabledBucketName = _resourcePrefix + "-lock-enabled";
        retentionAfterCreationBucketName = _resourcePrefix + "-retention-after-
creation";

        bucketNames.Add(noLockBucketName);
        bucketNames.Add(lockEnabledBucketName);
        bucketNames.Add(retentionAfterCreationBucketName);
    }

    // <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Setup(bool interactive)
    {
        Console.WriteLine(
            "\nFor this workflow, we will use the AWS SDK for .NET to create several
 S3\n" +
            "buckets and files to demonstrate working with S3 locking features.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you are ready to start.");
```

```
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nS3 buckets can be created either with or without object
lock enabled.");
        await _s3ActionsWrapper.CreateBucketWithObjectLock(noLockBucketName, false);
        await _s3ActionsWrapper.CreateBucketWithObjectLock(lockEnabledBucketName,
true);
        await
_s3ActionsWrapper.CreateBucketWithObjectLock(retentionAfterCreationBucketName,
false);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nA bucket can be configured to use object locking with a
default retention period.");
        await
_s3ActionsWrapper.ModifyBucketDefaultRetention(retentionAfterCreationBucketName,
true,
            ObjectLockRetentionMode.Governance, DateTime.UtcNow.AddDays(1));

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nObject lock policies can also be added to existing
buckets.");
        await _s3ActionsWrapper.EnableObjectLockOnBucket(lockEnabledBucketName);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        // Upload some files to the buckets.
        Console.WriteLine("\nNow let's add some test files:");
        var fileName = _configuration["exampleFileName"] ?? "exampleFile.txt";
        int fileCount = 2;
        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
```

```
                "This is a sample file for uploading to a bucket.");
        }

        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileCount; i++)
            {
                var numberedFileName = Path.GetFileNameWithoutExtension(fileName) +
i + Path.GetExtension(fileName);
                fileNames.Add(numberedFileName);
                await _s3ActionsWrapper.UploadFileAsync(bucketName,
numberedFileName, fileName);
            }
        }
        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        if (!interactive)
            return true;
        Console.WriteLine("\nNow we can set some object lock policies on individual
files:");
        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileNames.Count; i++)
            {
                // No modifications to the objects in the first bucket.
                if (bucketName != bucketNames[0])
                {
                    var exampleFileName = fileNames[i];
                    switch (i)
                    {
                        case 0:
                            {
                                var question =
                                    $"\nWould you like to add a legal hold to
{exampleFileName} in {bucketName}? (y/n)";
                                if (GetYesNoResponse(question))
                                {
                                    // Set a legal hold.
                                    await
_s3ActionsWrapper.ModifyObjectLegalHold(bucketName, exampleFileName,
ObjectLockLegalHoldStatus.On);
```

```
                                }
                                break;
                            }
                        case 1:
                            {
                                var question =
                                    $"\nWould you like to add a 1 day Governance
retention period to {exampleFileName} in {bucketName}? (y/n)" +
                                    "\nReminder: Only a user with the
s3:BypassGovernanceRetention permission will be able to delete this file or its
bucket until the retention period has expired.";
                                if (GetYesNoResponse(question))
                                {
                                    // Set a Governance mode retention period for 1
day.
                                    await
_s3ActionsWrapper.ModifyObjectRetentionPeriod(
                                        bucketName, exampleFileName,
                                        ObjectLockRetentionMode.Governance,
                                        DateTime.UtcNow.AddDays(1));
                                }
                                break;
                            }
                    }
                }
            }
        }
        Console.WriteLine(new string('-', 80));
        return true;
    }

    // <summary>
    /// List all of the current buckets and objects.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>The list of buckets and objects.</returns>
    public static async Task<List<S3ObjectVersion>> ListBucketsAndObjects(bool
interactive)
    {
        var allObjects = new List<S3ObjectVersion>();
        foreach (var bucketName in bucketNames)
        {
            var objectsInBucket = await
_s3ActionsWrapper.ListBucketObjectsAndVersions(bucketName);
```

```
            foreach (var objectKey in objectsInBucket.Versions)
            {
                allObjects.Add(objectKey);
            }
        }

        if (interactive)
        {
            Console.WriteLine("\nCurrent buckets and objects:\n");
            int i = 0;
            foreach (var bucketObject in allObjects)
            {
                i++;
                Console.WriteLine(
                    $"{i}: {bucketObject.Key} \n\tBucket:
{bucketObject.BucketName}\n\tVersion: {bucketObject.VersionId}");
            }
        }

        return allObjects;
    }

    /// <summary>
    /// Present the user with the demo action choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<bool> DemoActionChoices()
    {
        var choices = new string[]{
            "List all files in buckets.",
            "Attempt to delete a file.",
            "Attempt to delete a file with retention period bypass.",
            "Attempt to overwrite a file.",
            "View the object and bucket retention settings for a file.",
            "View the legal hold settings for a file.",
            "Finish the workflow."};

        var choice = 0;
        // Keep asking the user until they choose to move on.
        while (choice != 6)
        {
            Console.WriteLine(new string('-', 80));
            choice = GetChoiceResponse(
```

```
                "\nExplore the S3 locking features by selecting one of the following
choices:"
                , choices);
            Console.WriteLine(new string('-', 80));
            switch (choice)
            {
                case 0:
                    {
                        await ListBucketsAndObjects(true);
                        break;
                    }
                case 1:
                    {
                        Console.WriteLine("\nEnter the number of the object to
delete:");
                        var allFiles = await ListBucketsAndObjects(true);
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, false, allFiles[fileChoice].VersionId);
                        break;
                    }
                case 2:
                    {
                        Console.WriteLine("\nEnter the number of the object to
delete:");
                        var allFiles = await ListBucketsAndObjects(true);
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, true, allFiles[fileChoice].VersionId);
                        break;
                    }
                case 3:
                    {
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object to
overwrite:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        // Create the file if it does not already exist.
                        if (!File.Exists(allFiles[fileChoice].Key))
```

```
                        {
                            await using StreamWriter sw =
File.CreateText(allFiles[fileChoice].Key);
                            await sw.WriteLineAsync(
                                "This is a sample file for uploading to a bucket.");
                        }
                        await
_s3ActionsWrapper.UploadFileAsync(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, allFiles[fileChoice].Key);
                        break;
                    }
                case 4:
                    {
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object and
bucket to view:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.GetObjectRetention(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
                        await
_s3ActionsWrapper.GetBucketObjectLockConfiguration(allFiles[fileChoice].BucketName);
                        break;
                    }
                case 5:
                    {
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object to
view:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.GetObjectLegalHold(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
                        break;
                    }
            }
        }
        return true;
    }

    // <summary>
    /// Clean up the resources from the scenario.
```

```
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Cleanup(bool interactive)
    {
        Console.WriteLine(new string('-', 80));

        if (!interactive || GetYesNoResponse("Do you want to clean up all files and
buckets? (y/n) "))
        {
            // Remove all locks and delete all buckets and objects.
            var allFiles = await ListBucketsAndObjects(false);
            foreach (var fileInfo in allFiles)
            {
                // Check for a legal hold.
                var legalHold = await
_s3ActionsWrapper.GetObjectLegalHold(fileInfo.BucketName, fileInfo.Key);
                if (legalHold?.Status?.Value == ObjectLockLegalHoldStatus.On)
                {
                    await
_s3ActionsWrapper.ModifyObjectLegalHold(fileInfo.BucketName, fileInfo.Key,
ObjectLockLegalHoldStatus.Off);
                }

                // Check for a retention period.
                var retention = await
_s3ActionsWrapper.GetObjectRetention(fileInfo.BucketName, fileInfo.Key);
                var hasRetentionPeriod = retention?.Mode ==
ObjectLockRetentionMode.Governance && retention.RetainUntilDate >
DateTime.UtcNow.Date;
                await _s3ActionsWrapper.DeleteObjectFromBucket(fileInfo.BucketName,
fileInfo.Key, hasRetentionPeriod, fileInfo.VersionId);
            }

            foreach (var bucketName in bucketNames)
            {
                await _s3ActionsWrapper.DeleteBucketByName(bucketName);
            }

        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
```

```
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Helper method to get a choice response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="choices">The choices to print on the console.</param>
    /// <returns>The index of the selected choice</returns>
    private static int GetChoiceResponse(string? question, string[] choices)
    {
        if (question != null)
        {
            Console.WriteLine(question);

            for (int i = 0; i < choices.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {choices[i]}");
            }
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > choices.Length)
        {
```

```
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        return choiceNumber - 1;
    }
}
```

Una clase contenedora para funciones de S3.

```
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

namespace S3ObjectLockScenario;

/// <summary>
/// Encapsulate the Amazon S3 operations.
/// </summary>
public class S3ActionsWrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Constructor for the S3ActionsWrapper.
    /// </summary>
    /// <param name="amazonS3">The injected S3 client.</param>
    public S3ActionsWrapper(IAmazonS3 amazonS3, IConfiguration configuration)
    {
        _amazonS3 = amazonS3;
    }

    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
```

```
   public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
enableObjectLock)
   {
       Console.WriteLine($"\tCreating bucket {bucketName} with object lock
{enableObjectLock}.");
       try
       {
           var request = new PutBucketRequest
           {
               BucketName = bucketName,
               UseClientRegion = true,
               ObjectLockEnabledForBucket = enableObjectLock,
           };

           var response = await _amazonS3.PutBucketAsync(request);

           return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
       }
       catch (AmazonS3Exception ex)
       {
           Console.WriteLine($"Error creating bucket: '{ex.Message}'");
           return false;
       }
   }

   /// <summary>
   /// Enable object lock on an existing bucket.
   /// </summary>
   /// <param name="bucketName">The name of the bucket to modify.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> EnableObjectLockOnBucket(string bucketName)
   {
       try
       {
           // First, enable Versioning on the bucket.
           await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
           {
               BucketName = bucketName,
               VersioningConfig = new S3BucketVersioningConfig()
               {
                   EnableMfaDelete = false,
                   Status = VersionStatus.Enabled
               }
```

```
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
                },
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded an object lock policy to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date retention expires.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
        string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
    {
        try
        {
            var request = new PutObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Retention = new ObjectLockRetention()
                {
                    Mode = retention,
```

```
                    RetainUntilDate = retainUntilDate
                }
            };

            var response = await _amazonS3.PutObjectRetentionAsync(request);
            Console.WriteLine($"\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying retention period:
'{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to modify.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date for retention until.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
    {
        var enabledString = enableObjectLock ? "Enabled" : "Disabled";
        var timeDifference = retainUntilDate.Subtract(DateTime.Now);
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
```

```
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                    Rule = new ObjectLockRule()
                    {
                        DefaultRetention = new DefaultRetention()
                        {
                            Mode = retention,
                            Days = timeDifference.Days // Can be specified in days
or years but not both.
                        }
                    }
                }
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded a default retention to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying object lock: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Get the retention period for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object retention details.</returns>
    public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey
```

```
            };

            var response = await _amazonS3.GetObjectRetentionAsync(request);
            Console.WriteLine($"\tObject retention for {objectKey} in {bucketName}:
" +
                              $"\n\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
            return response.Retention;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock retention:
'{ex.Message}'");
            return new ObjectLockRetention();
        }
    }

    /// <summary>
    /// Set or modify a legal hold on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="holdStatus">The On or Off status for the legal hold.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectLegalHold(string bucketName,
        string objectKey, ObjectLockLegalHoldStatus holdStatus)
    {
        try
        {
            var request = new PutObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                LegalHold = new ObjectLockLegalHold()
                {
                    Status = holdStatus
                }
            };

            var response = await _amazonS3.PutObjectLegalHoldAsync(request);
            Console.WriteLine($"\tModified legal hold for {objectKey} in
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying legal hold: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }

    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>
    /// <returns>The bucket's object lock configuration details.</returns>
    public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
```

```
    {
        try
        {
            var request = new GetObjectLockConfigurationRequest()
            {
                BucketName = bucketName
            };

            var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tBucket object lock config for {bucketName} in
{bucketName}: " +
                              $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                              $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

            return response.ObjectLockConfiguration;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock config:
'{ex.Message}'");
            return new ObjectLockConfiguration();
        }
    }

    /// <summary>
    /// Upload a file from the local computer to an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object to
upload.</param>
    /// <returns>True if success.<returns>
    public async Task<bool> UploadFileAsync(string bucketName, string objectName,
string filePath)
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
            ChecksumAlgorithm = ChecksumAlgorithm.SHA256
        };
```

```
        var response = await _amazonS3.PutObjectAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"\tSuccessfully uploaded {objectName} to
 {bucketName}.");
            return true;
        }
        else
        {
            Console.WriteLine($"\tCould not upload {objectName} to {bucketName}.");
            return false;
        }
    }

    /// <summary>
    /// List bucket objects and versions.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <returns>The list of objects and versions.</returns>
    public async Task<ListVersionsResponse> ListBucketObjectsAndVersions(string
 bucketName)
    {
        var request = new ListVersionsRequest()
        {
            BucketName = bucketName
        };

        var response = await _amazonS3.ListVersionsAsync(request);
        return response;
    }

    /// <summary>
    /// Delete an object from a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="hasRetention">True if the object has retention settings.</
param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteObjectFromBucket(string bucketName, string
 objectKey, bool hasRetention, string? versionId = null)
    {
```

```
        try
        {
            var request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                VersionId = versionId,
            };
            if (hasRetention)
            {
                // Set the BypassGovernanceRetention header
                // if the file has retention settings.
                request.BypassGovernanceRetention = true;
            }
            await _amazonS3.DeleteObjectAsync(request);
            Console.WriteLine(
                $"Deleted {objectKey} in {bucketName}.");
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to delete object {objectKey} in bucket
{bucketName}: " + ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Delete a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteBucketByName(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest() { BucketName = bucketName, };
            var response = await _amazonS3.DeleteBucketAsync(request);
            Console.WriteLine($"\tDelete for {bucketName} complete.");
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
```

```
        {
            Console.WriteLine($"\tUnable to delete bucket {bucketName}: " +
 ex.Message);
            return false;
        }

    }

}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

Administrar listas de control de acceso (ACLs)

El siguiente ejemplo de código muestra cómo gestionar las listas de control de acceso (ACLs) para los buckets de Amazon S3.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;
```

```
    /// <summary>
    /// This example shows how to manage Amazon Simple Storage Service
    /// (Amazon S3) access control lists (ACLs) to control Amazon S3 bucket
    /// access.
    /// </summary>
    public class ManageACLs
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket1";
            string newBucketName = "doc-example-bucket2";
            string keyName = "sample-object.txt";
            string emailAddress = "someone@example.com";

            // If the AWS Region where your bucket is located is different from
            // the Region defined for the default user, pass the Amazon S3 bucket's
            // name to the client constructor. It should look like this:
            // RegionEndpoint bucketRegion = RegionEndpoint.USEast1;
            IAmazonS3 client = new AmazonS3Client();

            await TestBucketObjectACLsAsync(client, bucketName, newBucketName,
keyName, emailAddress);
        }

        /// <summary>
        /// Creates a new Amazon S3 bucket with a canned ACL, then retrieves the ACL
        /// information and then adds a new ACL to one of the objects in the
        /// Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// methods to create a bucket, get an ACL, and add a different ACL to
        /// one of the objects.</param>
        /// <param name="bucketName">A string representing the original Amazon S3
        /// bucket name.</param>
        /// <param name="newBucketName">A string representing the name of the
        /// new bucket that will be created.</param>
        /// <param name="keyName">A string representing the key name of an Amazon S3
        /// object for which we will change the ACL.</param>
        /// <param name="emailAddress">A string representing the email address
        /// belonging to the person to whom access to the Amazon S3 bucket will be
        /// granted.</param>
        public static async Task TestBucketObjectACLsAsync(
            IAmazonS3 client,
```

```
            string bucketName,
            string newBucketName,
            string keyName,
            string emailAddress)
        {
            try
            {
                // Create a new Amazon S3 bucket and specify canned ACL.
                var success = await CreateBucketWithCannedACLAsync(client,
 newBucketName);

                // Get the ACL on a bucket.
                await GetBucketACLAsync(client, bucketName);

                // Add (replace) the ACL on an object in a bucket.
                await AddACLToExistingObjectAsync(client, bucketName, keyName,
 emailAddress);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine($"Exception: {amazonS3Exception.Message}");
            }
        }

        /// <summary>
        /// Creates a new Amazon S3 bucket with a canned ACL attached.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// PutBucketAsync.</param>
        /// <param name="newBucketName">A string representing the name of the
        /// new Amazon S3 bucket.</param>
        /// <returns>Returns a boolean value indicating success or failure.</
returns>
        public static async Task<bool> CreateBucketWithCannedACLAsync(IAmazonS3
 client, string newBucketName)
        {
            var request = new PutBucketRequest()
            {
                BucketName = newBucketName,
                BucketRegion = S3Region.EUWest1,

                // Add a canned ACL.
                CannedACL = S3CannedACL.LogDeliveryWrite,
            };
```

```
            var response = await client.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Retrieves the ACL associated with the Amazon S3 bucket name in the
        /// bucketName parameter.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// PutBucketAsync.</param>
        /// <param name="bucketName">The Amazon S3 bucket for which we want to get
 the
        /// ACL list.</param>
        /// <returns>Returns an S3AccessControlList returned from the call to
        /// GetACLAsync.</returns>
        public static async Task<S3AccessControlList> GetBucketACLAsync(IAmazonS3
 client, string bucketName)
        {
            GetACLResponse response = await client.GetACLAsync(new GetACLRequest
            {
                BucketName = bucketName,
            });

            return response.AccessControlList;
        }



        /// <summary>
        /// Adds a new ACL to an existing object in the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// PutBucketAsync.</param>
        /// <param name="bucketName">A string representing the name of the Amazon S3
        /// bucket containing the object to which we want to apply a new ACL.</
param>
        /// <param name="keyName">A string representing the name of the object
        /// to which we want to apply the new ACL.</param>
        /// <param name="emailAddress">The email address of the person to whom
        /// we will be applying to whom access will be granted.</param>
        public static async Task AddACLToExistingObjectAsync(IAmazonS3 client,
 string bucketName, string keyName, string emailAddress)
```

```
        {
            // Retrieve the ACL for an object.
            GetACLResponse aclResponse = await client.GetACLAsync(new GetACLRequest
            {
                BucketName = bucketName,
                Key = keyName,
            });

            S3AccessControlList acl = aclResponse.AccessControlList;

            // Retrieve the owner.
            Owner owner = acl.Owner;

            // Clear existing grants.
            acl.Grants.Clear();

            // Add a grant to reset the owner's full permission
            // (the previous clear statement removed all permissions).
            var fullControlGrant = new S3Grant
            {
                Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id },
            };
            acl.AddGrant(fullControlGrant.Grantee, S3Permission.FULL_CONTROL);

            // Specify email to identify grantee for granting permissions.
            var grantUsingEmail = new S3Grant
            {
                Grantee = new S3Grantee { EmailAddress = emailAddress },
                Permission = S3Permission.WRITE_ACP,
            };

            // Specify log delivery group as grantee.
            var grantLogDeliveryGroup = new S3Grant
            {
                Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" },
                Permission = S3Permission.WRITE,
            };

            // Create a new ACL.
            var newAcl = new S3AccessControlList
            {
                Grants = new List<S3Grant> { grantUsingEmail,
 grantLogDeliveryGroup },
```

```
                Owner = owner,
            };

            // Set the new ACL. We're throwing away the response here.
            _ = await client.PutACLAsync(new PutACLRequest
            {
                BucketName = bucketName,
                Key = keyName,
                AccessControlList = newAcl,
            });
        }

    }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - GetBucketAcl
  - GetObjectAcl
  - PutBucketAcl
  - PutObjectAcl

Ejecución de una copia multiparte

El siguiente ejemplo de código muestra cómo realizar una copia multiparte de un objeto de Amazon S3.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
```

```
using Amazon.S3.Model;

/// <summary>
/// This example shows how to perform a multi-part copy from one Amazon
/// Simple Storage Service (Amazon S3) bucket to another.
/// </summary>
public class MPUapiCopyObj
{
    private const string SourceBucket = "doc-example-bucket1";
    private const string TargetBucket = "doc-example-bucket2";
    private const string SourceObjectKey = "example.mov";
    private const string TargetObjectKey = "copied_video_file.mov";

    /// <summary>
    /// This method starts the multi-part upload.
    /// </summary>
    public static async Task Main()
    {
        var s3Client = new AmazonS3Client();
        Console.WriteLine("Copying object...");
        await MPUCopyObjectAsync(s3Client);
    }

    /// <summary>
    /// This method uses the passed client object to perform a multipart
    /// copy operation.
    /// </summary>
    /// <param name="client">An Amazon S3 client object that will be used
    /// to perform the copy.</param>
    public static async Task MPUCopyObjectAsync(AmazonS3Client client)
    {
        // Create a list to store the copy part responses.
        var copyResponses = new List<CopyPartResponse>();

        // Setup information required to initiate the multipart upload.
        var initiateRequest = new InitiateMultipartUploadRequest
        {
            BucketName = TargetBucket,
            Key = TargetObjectKey,
        };

        // Initiate the upload.
        InitiateMultipartUploadResponse initResponse =
            await client.InitiateMultipartUploadAsync(initiateRequest);
```

```
            // Save the upload ID.
            string uploadId = initResponse.UploadId;

            try
            {
                // Get the size of the object.
                var metadataRequest = new GetObjectMetadataRequest
                {
                    BucketName = SourceBucket,
                    Key = SourceObjectKey,
                };

                GetObjectMetadataResponse metadataResponse =
                    await client.GetObjectMetadataAsync(metadataRequest);
                var objectSize = metadataResponse.ContentLength; // Length in bytes.

                // Copy the parts.
                var partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

                long bytePosition = 0;
                for (int i = 1; bytePosition < objectSize; i++)
                {
                    var copyRequest = new CopyPartRequest
                    {
                        DestinationBucket = TargetBucket,
                        DestinationKey = TargetObjectKey,
                        SourceBucket = SourceBucket,
                        SourceKey = SourceObjectKey,
                        UploadId = uploadId,
                        FirstByte = bytePosition,
                        LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
                        PartNumber = i,
                    };

                    copyResponses.Add(await client.CopyPartAsync(copyRequest));

                    bytePosition += partSize;
                }

                // Set up to complete the copy.
                var completeRequest = new CompleteMultipartUploadRequest
                {
```

```
                    BucketName = TargetBucket,
                    Key = TargetObjectKey,
                    UploadId = initResponse.UploadId,
                };
                completeRequest.AddPartETags(copyResponses);

                // Complete the copy.
                CompleteMultipartUploadResponse completeUploadResponse =
                    await client.CompleteMultipartUploadAsync(completeRequest);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine($"Error encountered on server.
Message:'{e.Message}' when writing an object");
            }
            catch (Exception e)
            {
                Console.WriteLine($"Unknown encountered on server.
Message:'{e.Message}' when writing an object");
            }
        }
    }
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CompleteMultipartUpload
  - CreateMultipartUpload
  - GetObjectMetadata
  - UploadPartCopy

Transformación de datos con S3 Object Lambda

En el siguiente ejemplo de código se muestra cómo transformar datos para su aplicación con S3 Object Lambda.

AWS SDK for .NET

Muestra cómo añadir código personalizado a GET las solicitudes estándar de S3 para modificar el objeto solicitado recuperado de S3 de forma que se adapte a las necesidades del cliente o la aplicación solicitante.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Lambda
- Amazon S3

Cargar o descargar archivos grandes

En el siguiente ejemplo de código se muestra cómo cargar o descargar archivos grandes en y desde Amazon S3.

Para obtener información, consulte Carga de un objeto con carga multiparte.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Llame a funciones que transfieran archivos hacia y desde un bucket de S3 mediante Amazon S3 TransferUtility.

```
global using System.Text;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.S3.Transfer;
global using TransferUtilityBasics;



// This Amazon S3 client uses the default user credentials
// defined for this computer.
```

```
using Microsoft.Extensions.Configuration;

IAmazonS3 client = new AmazonS3Client();
var transferUtil = new TransferUtility(client);
IConfiguration _configuration;

_configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load test settings from JSON file.
    .AddJsonFile("settings.local.json",
        true) // Optionally load local settings.
    .Build();

// Edit the values in settings.json to use an S3 bucket and files that
// exist on your AWS account and on the local computer where you
// run this scenario.
var bucketName = _configuration["BucketName"];
var localPath =
 $"{Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)}\
\TransferFolder";

DisplayInstructions();

PressEnter();

Console.WriteLine();

// Upload a single file to an S3 bucket.
DisplayTitle("Upload a single file");

var fileToUpload = _configuration["FileToUpload"];
Console.WriteLine($"Uploading {fileToUpload} to the S3 bucket, {bucketName}.");

var success = await TransferMethods.UploadSingleFileAsync(transferUtil, bucketName,
 fileToUpload, localPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the file, {fileToUpload} to
 {bucketName}.");
}

PressEnter();

// Upload a local directory to an S3 bucket.
```

```
DisplayTitle("Upload all files from a local directory");
Console.WriteLine("Upload all the files in a local folder to an S3 bucket.");
const string keyPrefix = "UploadFolder";
var uploadPath = $"{localPath}\\UploadFolder";

Console.WriteLine($"Uploading the files in {uploadPath} to {bucketName}");
DisplayTitle($"{uploadPath} files");
DisplayLocalFiles(uploadPath);
Console.WriteLine();

PressEnter();

success = await TransferMethods.UploadFullDirectoryAsync(transferUtil, bucketName,
 keyPrefix, uploadPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the files in {uploadPath} to
 {bucketName}.");
    Console.WriteLine($"{bucketName} currently contains the following files:");
    await DisplayBucketFiles(client, bucketName, keyPrefix);
    Console.WriteLine();
}

PressEnter();

// Download a single file from an S3 bucket.
DisplayTitle("Download a single file");
Console.WriteLine("Now we will download a single file from an S3 bucket.");

var keyName = _configuration["FileToDownload"];

Console.WriteLine($"Downloading {keyName} from {bucketName}.");

success = await TransferMethods.DownloadSingleFileAsync(transferUtil, bucketName,
 keyName, localPath);
if (success)
{
    Console.WriteLine("$Successfully downloaded the file, {keyName} from
 {bucketName}.");
}

PressEnter();

// Download the contents of a directory from an S3 bucket.
```

```
DisplayTitle("Download the contents of an S3 bucket");
var s3Path = _configuration["S3Path"];
var downloadPath = $"{localPath}\\{s3Path}";

Console.WriteLine($"Downloading the contents of {bucketName}\\{s3Path}");
Console.WriteLine($"{bucketName}\\{s3Path} contains the following files:");
await DisplayBucketFiles(client, bucketName, s3Path);
Console.WriteLine();

success = await TransferMethods.DownloadS3DirectoryAsync(transferUtil, bucketName,
 s3Path, downloadPath);
if (success)
{
    Console.WriteLine($"Downloaded the files in {bucketName} to {downloadPath}.");
    Console.WriteLine($"{downloadPath} now contains the following files:");
    DisplayLocalFiles(downloadPath);
}

Console.WriteLine("\nThe TransferUtility Basics application has completed.");
PressEnter();

// Displays the title for a section of the scenario.
static void DisplayTitle(string titleText)
{
    var sepBar = new string('-', Console.WindowWidth);

    Console.WriteLine(sepBar);
    Console.WriteLine(CenterText(titleText));
    Console.WriteLine(sepBar);
}

// Displays a description of the actions to be performed by the scenario.
static void DisplayInstructions()
{
    var sepBar = new string('-', Console.WindowWidth);

    DisplayTitle("Amazon S3 Transfer Utility Basics");
    Console.WriteLine("This program shows how to use the Amazon S3 Transfer
 Utility.");
    Console.WriteLine("It performs the following actions:");
    Console.WriteLine("\t1. Upload a single object to an S3 bucket.");
    Console.WriteLine("\t2. Upload an entire directory from the local computer to an
\n\t  S3 bucket.");
    Console.WriteLine("\t3. Download a single object from an S3 bucket.");
```

```
    Console.WriteLine("\t4. Download the objects in an S3 bucket to a local
 directory.");
    Console.WriteLine($"\n{sepBar}");
}

// Pauses the scenario.
static void PressEnter()
{
    Console.WriteLine("Press <Enter> to continue.");
    _ = Console.ReadLine();
    Console.WriteLine("\n");
}

// Returns the string textToCenter, padded on the left with spaces
// that center the text on the console display.
static string CenterText(string textToCenter)
{
    var centeredText = new StringBuilder();
    var screenWidth = Console.WindowWidth;
    centeredText.Append(new string(' ', (int)(screenWidth - textToCenter.Length) /
 2));
    centeredText.Append(textToCenter);
    return centeredText.ToString();
}

// Displays a list of file names included in the specified path.
static void DisplayLocalFiles(string localPath)
{
    var fileList = Directory.GetFiles(localPath);
    if (fileList.Length > 0)
    {
        foreach (var fileName in fileList)
        {
            Console.WriteLine(fileName);
        }
    }
}

// Displays a list of the files in the specified S3 bucket and prefix.
static async Task DisplayBucketFiles(IAmazonS3 client, string bucketName, string
 s3Path)
{
    ListObjectsV2Request request = new()
    {
```

```
        BucketName = bucketName,
        Prefix = s3Path,
        MaxKeys = 5,
    };

    var response = new ListObjectsV2Response();

    do
    {
        response = await client.ListObjectsV2Async(request);

        response.S3Objects
            .ForEach(obj => Console.WriteLine($"{obj.Key}"));

        // If the response is truncated, set the request ContinuationToken
        // from the NextContinuationToken property of the response.
        request.ContinuationToken = response.NextContinuationToken;
    } while (response.IsTruncated);
}
```

Cargar un solo archivo.

```
        /// <summary>
        /// Uploads a single file from the local computer to an S3 bucket.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the file
        /// will be stored.</param>
        /// <param name="fileName">The name of the file to upload.</param>
        /// <param name="localPath">The local path where the file is stored.</param>
        /// <returns>A boolean value indicating the success of the action.</returns>
        public static async Task<bool> UploadSingleFileAsync(
            TransferUtility transferUtil,
            string bucketName,
            string fileName,
            string localPath)
        {
            if (File.Exists($"{localPath}\\{fileName}"))
            {
```

```
                try
                {
                    await transferUtil.UploadAsync(new TransferUtilityUploadRequest
                    {
                        BucketName = bucketName,
                        Key = fileName,
                        FilePath = $"{localPath}\\{fileName}",
                    });

                    return true;
                }
                catch (AmazonS3Exception s3Ex)
                {
                    Console.WriteLine($"Could not upload {fileName} from {localPath}
because:");
                    Console.WriteLine(s3Ex.Message);
                    return false;
                }
            }
            else
            {
                Console.WriteLine($"{fileName} does not exist in {localPath}");
                return false;
            }
        }
```

Cargar un directorio local completo.

```
        /// <summary>
        /// Uploads all the files in a local directory to a directory in an S3
        /// bucket.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the files
        /// will be stored.</param>
        /// <param name="keyPrefix">The key prefix is the S3 directory where
        /// the files will be stored.</param>
        /// <param name="localPath">The local directory that contains the files
        /// to be uploaded.</param>
```

```
        /// <returns>A Boolean value representing the success of the action.</
returns>
        public static async Task<bool> UploadFullDirectoryAsync(
            TransferUtility transferUtil,
            string bucketName,
            string keyPrefix,
            string localPath)
        {
            if (Directory.Exists(localPath))
            {
                try
                {
                    await transferUtil.UploadDirectoryAsync(new
 TransferUtilityUploadDirectoryRequest
                    {
                        BucketName = bucketName,
                        KeyPrefix = keyPrefix,
                        Directory = localPath,
                    });

                    return true;
                }
                catch (AmazonS3Exception s3Ex)
                {
                    Console.WriteLine($"Can't upload the contents of {localPath}
 because:");
                    Console.WriteLine(s3Ex?.Message);
                    return false;
                }
            }
            else
            {
                Console.WriteLine($"The directory {localPath} does not exist.");
                return false;
            }
        }
```

Descargar un solo archivo.

```
        /// <summary>
```

```
        /// Download a single file from an S3 bucket to the local computer.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket containing the
        /// file to download.</param>
        /// <param name="keyName">The name of the file to download.</param>
        /// <param name="localPath">The path on the local computer where the
        /// downloaded file will be saved.</param>
        /// <returns>A Boolean value indicating the results of the action.</returns>
        public static async Task<bool> DownloadSingleFileAsync(
        TransferUtility transferUtil,
            string bucketName,
            string keyName,
            string localPath)
        {
            await transferUtil.DownloadAsync(new TransferUtilityDownloadRequest
            {
                BucketName = bucketName,
                Key = keyName,
                FilePath = $"{localPath}\\{keyName}",
            });

            return (File.Exists($"{localPath}\\{keyName}"));
        }
```

Descargar el contenido de un bucket de S3.

```
        /// <summary>
        /// Downloads the contents of a directory in an S3 bucket to a
        /// directory on the local computer.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The bucket containing the files to download.</
param>
        /// <param name="s3Path">The S3 directory where the files are located.</
param>
        /// <param name="localPath">The local path to which the files will be
        /// saved.</param>
```

```
        /// <returns>A Boolean value representing the success of the action.</
returns>
        public static async Task<bool> DownloadS3DirectoryAsync(
            TransferUtility transferUtil,
            string bucketName,
            string s3Path,
            string localPath)
        {
            int fileCount = 0;

            // If the directory doesn't exist, it will be created.
            if (Directory.Exists(s3Path))
            {
                var files = Directory.GetFiles(localPath);
                fileCount = files.Length;
            }

            await transferUtil.DownloadDirectoryAsync(new
 TransferUtilityDownloadDirectoryRequest
            {
                BucketName = bucketName,
                LocalDirectory = localPath,
                S3Directory = s3Path,
            });

            if (Directory.Exists(localPath))
            {
                var files = Directory.GetFiles(localPath);
                if (files.Length > fileCount)
                {
                    return true;
                }

                // No change in the number of files. Assume
                // the download failed.
                return false;
            }

            // The local directory doesn't exist. No files
            // were downloaded.
            return false;
        }
```

Realice un seguimiento del progreso de una carga mediante TransferUtility.

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to track the progress of a multipart upload
/// using the Amazon Simple Storage Service (Amazon S3) TransferUtility to
/// upload to an Amazon S3 bucket.
/// </summary>
public class TrackMPUUsingHighLevelAPI
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "sample_pic.png";
        string path = "filepath/directory/";
        string filePath = $"{path}{keyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        await TrackMPUAsync(client, bucketName, filePath, keyName);
    }

    /// <summary>
    /// Starts an Amazon S3 multipart upload and assigns an event handler to
    /// track the progress of the upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// perform the multipart upload.</param>
    /// <param name="bucketName">The name of the bucket to which to upload
    /// the file.</param>
    /// <param name="filePath">The path, including the file name of the
    /// file to be uploaded to the Amazon S3 bucket.</param>
    /// <param name="keyName">The file name to be used in the
```

```
        /// destination Amazon S3 bucket.</param>
        public static async Task TrackMPUAsync(
            IAmazonS3 client,
            string bucketName,
            string filePath,
            string keyName)
        {
            try
            {
                var fileTransferUtility = new TransferUtility(client);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                var uploadRequest =
                    new TransferUtilityUploadRequest
                    {
                        BucketName = bucketName,
                        FilePath = filePath,
                        Key = keyName,
                    };

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>(
                        UploadRequest_UploadPartProgressEvent);

                await fileTransferUtility.UploadAsync(uploadRequest);
                Console.WriteLine("Upload completed");
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error:: {ex.Message}");
            }
        }

        /// <summary>
        /// Event handler to check the progress of the multipart upload.
        /// </summary>
        /// <param name="sender">The object that raised the event.</param>
        /// <param name="e">The object that contains multipart upload
        /// information.</param>
        public static void UploadRequest_UploadPartProgressEvent(object sender,
UploadProgressArgs e)
        {
            // Process event.
```

```
            Console.WriteLine($"{e.TransferredBytes}/{e.TotalBytes}");
        }
    }
```

Cargar un objeto con cifrado.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Uses the Amazon Simple Storage Service (Amazon S3) low level API to
/// perform a multipart upload to an Amazon S3 bucket.
/// </summary>
public class SSECLowLevelMPUcopyObject
{
    public static async Task Main()
    {
        string existingBucketName = "doc-example-bucket";
        string sourceKeyName = "sample_file.txt";
        string targetKeyName = "sample_file_copy.txt";
        string filePath = $"sample\\{targetKeyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USEast1.
        IAmazonS3 client = new AmazonS3Client();

        // Create the encryption key.
        var base64Key = CreateEncryptionKey();

        await CreateSampleObjUsingClientEncryptionKeyAsync(
            client,
            existingBucketName,
            sourceKeyName,
            filePath,
```

```
                base64Key);
        }

        /// <summary>
        /// Creates the encryption key to use with the multipart upload.
        /// </summary>
        /// <returns>A string containing the base64-encoded key for encrypting
        /// the multipart upload.</returns>
        public static string CreateEncryptionKey()
        {
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);
            return base64Key;
        }

        /// <summary>
        /// Creates and uploads an object using a multipart upload.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 object used to
        /// initialize and perform the multipart upload.</param>
        /// <param name="existingBucketName">The name of the bucket to which
        /// the object will be uploaded.</param>
        /// <param name="sourceKeyName">The source object name.</param>
        /// <param name="filePath">The location of the source object.</param>
        /// <param name="base64Key">The encryption key to use with the upload.</
param>
        public static async Task CreateSampleObjUsingClientEncryptionKeyAsync(
            IAmazonS3 client,
            string existingBucketName,
            string sourceKeyName,
            string filePath,
            string base64Key)
        {
            List<UploadPartResponse> uploadResponses = new
 List<UploadPartResponse>();

            InitiateMultipartUploadRequest initiateRequest = new
 InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
```

```
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            InitiateMultipartUploadResponse initResponse =
               await client.InitiateMultipartUploadAsync(initiateRequest);

            long contentLength = new FileInfo(filePath).Length;
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

            try
            {
                long filePosition = 0;
                for (int i = 1; filePosition < contentLength; i++)
                {
                    UploadPartRequest uploadRequest = new UploadPartRequest
                    {
                        BucketName = existingBucketName,
                        Key = sourceKeyName,
                        UploadId = initResponse.UploadId,
                        PartNumber = i,
                        PartSize = partSize,
                        FilePosition = filePosition,
                        FilePath = filePath,
                        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                        ServerSideEncryptionCustomerProvidedKey = base64Key,
                    };

                    // Upload part and add response to our list.
                    uploadResponses.Add(await
client.UploadPartAsync(uploadRequest));

                    filePosition += partSize;
                }

                CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    UploadId = initResponse.UploadId,
                };
```

```
                completeRequest.AddPartETags(uploadResponses);

                CompleteMultipartUploadResponse completeUploadResponse =
                    await client.CompleteMultipartUploadAsync(completeRequest);
            }
            catch (Exception exception)
            {
                Console.WriteLine($"Exception occurred: {exception.Message}");

                // If there was an error, abort the multipart upload.
                AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    UploadId = initResponse.UploadId,
                };

                await client.AbortMultipartUploadAsync(abortMPURequest);
            }
        }
    }
```

Ejemplos sin servidor

Invocación de una función de Lambda desde un desencadenador de Amazon S3

En el siguiente ejemplo de código se muestra cómo implementar una función de Lambda que recibe un evento activado al cargar un objeto en un bucket de S3. La función recupera el nombre del bucket de S3 y la clave del objeto del parámetro de evento y llama a Amazon S3 API para recuperar y registrar el tipo de contenido del objeto.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un evento de S3 con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");
```

```
                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Ejemplos de S3 Glacier utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET con S3 Glacier.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Amazon S3 Glacier

En el siguiente ejemplo de código se muestra cómo empezar a utilizar Amazon S3 Glacier.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.Glacier;
using Amazon.Glacier.Model;

namespace GlacierActions;

public static class HelloGlacier
{
    static async Task Main()
    {
        var glacierService = new AmazonGlacierClient();

        Console.WriteLine("Hello Amazon Glacier!");
        Console.WriteLine("Let's list your Glacier vaults:");

        // You can use await and any of the async methods to get a response.
        // Let's get the vaults using a paginator.
        var glacierVaultPaginator = glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            Console.WriteLine($"{vault.CreationDate}:{vault.VaultName}, ARN:
{vault.VaultARN}");
        }
    }
}
```

- Para API obtener más información, consulte ListVaultsla AWS SDK for .NET APIReferencia.

Temas

- Acciones

Acciones

`AddTagsToVault`

En el siguiente ejemplo de código, se muestra cómo usar `AddTagsToVault`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Add tags to the items in an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to add tags to.</param>
    /// <param name="key">The name of the object to tag.</param>
    /// <param name="value">The tag value to add.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddTagsToVaultAsync(string vaultName, string key, string
value)
    {
        var request = new AddTagsToVaultRequest
        {
            Tags = new Dictionary<string, string>
                {
                    { key, value },
                },
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.AddTagsToVaultAsync(request);
        return response.HttpStatusCode == HttpStatusCode.NoContent;
    }
```

- Para API obtener más información, consulte AddTagsToVaultla AWS SDK for .NET APIReferencia.

`CreateVault`

En el siguiente ejemplo de código, se muestra cómo usar `CreateVault`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Create an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to create.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateVaultAsync(string vaultName)
{
    var request = new CreateVaultRequest
    {
        // Setting the AccountId to "-" means that
        // the account associated with the current
        // account will be used.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.CreateVaultAsync(request);

    Console.WriteLine($"Created {vaultName} at: {response.Location}");

    return response.HttpStatusCode == HttpStatusCode.Created;
}
```

- Para API obtener más información, consulte CreateVaultla AWS SDK for .NET APIReferencia.

`DescribeVault`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeVault`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Describe an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to describe.</param>
    /// <returns>The Amazon Resource Name (ARN) of the vault.</returns>
    public async Task<string> DescribeVaultAsync(string vaultName)
    {
        var request = new DescribeVaultRequest
        {
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.DescribeVaultAsync(request);

        // Display the information about the vault.
        Console.WriteLine($"{response.VaultName}\tARN: {response.VaultARN}");
        Console.WriteLine($"Created on: {response.CreationDate}\tNumber of Archives:
{response.NumberOfArchives}\tSize (in bytes): {response.SizeInBytes}");
        if (response.LastInventoryDate != DateTime.MinValue)
        {
            Console.WriteLine($"Last inventory: {response.LastInventoryDate}");
        }

        return response.VaultARN;
    }
```

- Para API obtener más información, consulte DescribeVaultla AWS SDK for .NET APIReferencia.

`InitiateJob`

En el siguiente ejemplo de código, se muestra cómo usar `InitiateJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recupera un archivo de una bóveda. En este ejemplo se usa la ArchiveTransferManager clase. Para API obtener más información, consulte ArchiveTransferManager.

```
   /// <summary>
   /// Download an archive from an Amazon S3 Glacier vault using the Archive
   /// Transfer Manager.
   /// </summary>
   /// <param name="vaultName">The name of the vault containing the object.</param>
   /// <param name="archiveId">The Id of the archive to download.</param>
   /// <param name="localFilePath">The local directory where the file will
   /// be stored after download.</param>
   /// <returns>Async Task.</returns>
   public async Task<bool> DownloadArchiveWithArchiveManagerAsync(string vaultName,
string archiveId, string localFilePath)
   {
       try
       {
           var manager = new ArchiveTransferManager(_glacierService);

           var options = new DownloadOptions
           {
               StreamTransferProgress = Progress!,
           };

           // Download an archive.
           Console.WriteLine("Initiating the archive retrieval job and then polling
SQS queue for the archive to be available.");
           Console.WriteLine("When the archive is available, downloading will
begin.");
           await manager.DownloadAsync(vaultName, archiveId, localFilePath,
options);
```

```
            return true;
        }
        catch (AmazonGlacierException ex)
        {
            Console.WriteLine(ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Event handler to track the progress of the Archive Transfer Manager.
    /// </summary>
    /// <param name="sender">The object that raised the event.</param>
    /// <param name="args">The argument values from the object that raised the
    /// event.</param>
    static void Progress(object sender, StreamTransferProgressArgs args)
    {
        if (args.PercentDone != _currentPercentage)
        {
            _currentPercentage = args.PercentDone;
            Console.WriteLine($"Downloaded {_currentPercentage}%");
        }
    }
```

- Para API obtener más información, consulte InitiateJobla AWS SDK for .NET APIReferencia.

`ListJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
```

```
    /// List Amazon S3 Glacier jobs.
    /// </summary>
    /// <param name="vaultName">The name of the vault to list jobs for.</param>
    /// <returns>A list of Amazon S3 Glacier jobs.</returns>
    public async Task<List<GlacierJobDescription>> ListJobsAsync(string vaultName)
    {
        var request = new ListJobsRequest
        {
            // Using a hyphen "-" for the Account Id will
            // cause the SDK to use the Account Id associated
            // with the current account.
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.ListJobsAsync(request);

        return response.JobList;
    }
```

- Para API obtener más información, consulte ListJobsla AWS SDK for .NET APIReferencia.

`ListTagsForVault`

En el siguiente ejemplo de código, se muestra cómo usar `ListTagsForVault`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List tags for an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to list tags for.</param>
    /// <returns>A dictionary listing the tags attached to each object in the
```

```
    /// vault and its tags.</returns>
    public async Task<Dictionary<string, string>> ListTagsForVaultAsync(string
vaultName)
    {
        var request = new ListTagsForVaultRequest
        {
            // Using a hyphen "-" for the Account Id will
            // cause the SDK to use the Account Id associated
            // with the default user.
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.ListTagsForVaultAsync(request);

        return response.Tags;
    }
```

- Para API obtener más información, consulte ListTagsForVaultla AWS SDK for .NET APIReferencia.

`ListVaults`

En el siguiente ejemplo de código, se muestra cómo usar `ListVaults`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List the Amazon S3 Glacier vaults associated with the current account.
    /// </summary>
    /// <returns>A list containing information about each vault.</returns>
    public async Task<List<DescribeVaultOutput>> ListVaultsAsync()
    {
```

```
        var glacierVaultPaginator = _glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });
        var vaultList = new List<DescribeVaultOutput>();

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            vaultList.Add(vault);
        }

        return vaultList;
    }
```

- Para API obtener más información, consulte ListVaultsla AWS SDK for .NET APIReferencia.

`UploadArchive`

En el siguiente ejemplo de código, se muestra cómo usar `UploadArchive`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Upload an object to an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the Amazon S3 Glacier vault to upload
    /// the archive to.</param>
    /// <param name="archiveFilePath">The file path of the archive to upload to the
vault.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<string> UploadArchiveWithArchiveManager(string vaultName,
string archiveFilePath)
    {
        try
        {
```

```
            var manager = new ArchiveTransferManager(_glacierService);

            // Upload an archive.
            var response = await manager.UploadAsync(vaultName, "upload archive
test", archiveFilePath);
            return response.ArchiveId;
        }
        catch (AmazonGlacierException ex)
        {
            Console.WriteLine(ex.Message);
            return string.Empty;
        }
    }
```

- Para API obtener más información, consulte UploadArchivela AWS SDK for .NET APIReferencia.

SageMaker ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with SageMaker.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola SageMaker

Los siguientes ejemplos de código muestran cómo empezar a usarlo SageMaker.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.SageMaker;
using Amazon.SageMaker.Model;

namespace SageMakerActions;

public static class HelloSageMaker
{
    static async Task Main(string[] args)
    {
        var sageMakerClient = new AmazonSageMakerClient();

        Console.WriteLine($"Hello Amazon SageMaker! Let's list some of your notebook
 instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five notebook instances.
        var response = await sageMakerClient.ListNotebookInstancesAsync(
            new ListNotebookInstancesRequest()
            {
                MaxResults = 5
            });

        if (!response.NotebookInstances.Any())
        {
            Console.WriteLine($"No notebook instances found.");
            Console.WriteLine("See https://docs.aws.amazon.com/sagemaker/latest/dg/
howitworks-create-ws.html to create one.");
        }

        foreach (var notebookInstance in response.NotebookInstances)
        {
```

```
            Console.WriteLine($"\tInstance:
 {notebookInstance.NotebookInstanceName}");
            Console.WriteLine($"\tArn: {notebookInstance.NotebookInstanceArn}");
            Console.WriteLine($"\tCreation Date:
 {notebookInstance.CreationTime.ToShortDateString()}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListNotebookInstancesla AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios

Acciones

`CreatePipeline`

En el siguiente ejemplo de código, se muestra cómo usar `CreatePipeline`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
 already exists.
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
 string name, string description, string displayName)
```

```
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });

            return createResponse.PipelineArn;
        }
    }
```

- Para API obtener más información, consulte CreatePipelinela AWS SDK for .NET APIReferencia.

`DeletePipeline`

En el siguiente ejemplo de código, se muestra cómo usar `DeletePipeline`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Delete a SageMaker pipeline by name.
/// </summary>
/// <param name="pipelineName">The name of the pipeline to delete.</param>
/// <returns>The ARN of the pipeline.</returns>
public async Task<string> DeletePipelineByName(string pipelineName)
{
    var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
        new DeletePipelineRequest()
        {
            PipelineName = pipelineName
        });

    return deleteResponse.PipelineArn;
}
```

- Para API obtener más información, consulte DeletePipelinela AWS SDK for .NET APIReferencia.

`DescribePipelineExecution`

En el siguiente ejemplo de código, se muestra cómo usar `DescribePipelineExecution`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Check the status of a run.
   /// </summary>
   /// <param name="pipelineExecutionArn">The ARN.</param>
   /// <returns>The status of the pipeline.</returns>
   public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
   {
       var describeResponse = await
_amazonSageMaker.DescribePipelineExecutionAsync(
           new DescribePipelineExecutionRequest()
           {
               PipelineExecutionArn = pipelineExecutionArn
           });

       return describeResponse.PipelineExecutionStatus;
   }
```

- Para API obtener más información, consulte DescribePipelineExecutionla AWS SDK for .NET APIReferencia.

`StartPipelineExecution`

En el siguiente ejemplo de código, se muestra cómo usar `StartPipelineExecution`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Run a pipeline with input and output file locations.
   /// </summary>
   /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
   /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
```

```
    /// <param name="outputLocationUrl">The output location in Amazon S3.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="executionRoleArn">The ARN of the role.</param>
    /// <returns>The ARN of the pipeline run.</returns>
    public async Task<string> ExecutePipeline(
        string queueUrl,
        string inputLocationUrl,
        string outputLocationUrl,
        string pipelineName,
        string executionRoleArn)
    {
        var inputConfig = new VectorEnrichmentJobInputConfig()
        {
            DataSourceConfig = new()
            {
                S3Data = new VectorEnrichmentJobS3Data()
                {
                    S3Uri = inputLocationUrl
                }
            },
            DocumentType = VectorEnrichmentJobDocumentType.CSV
        };

        var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = outputLocationUrl
            }
        };

        var jobConfig = new VectorEnrichmentJobConfig()
        {
            ReverseGeocodingConfig = new ReverseGeocodingConfig()
            {
                XAttributeName = "Longitude",
                YAttributeName = "Latitude"
            }
        };

#pragma warning disable SageMaker1002 // Property value does not match required
 pattern is allowed here to match the pipeline definition.
        var startExecutionResponse = await
 _amazonSageMaker.StartPipelineExecutionAsync(
```

```
            new StartPipelineExecutionRequest()
            {
                PipelineName = pipelineName,
                PipelineExecutionDisplayName = pipelineName + "-example-execution",
                PipelineParameters = new List<Parameter>()
                {
                    new Parameter() { Name = "parameter_execution_role", Value =
 executionRoleArn },
                    new Parameter() { Name = "parameter_queue_url", Value =
 queueUrl },
                    new Parameter() { Name = "parameter_vej_input_config", Value =
 JsonSerializer.Serialize(inputConfig) },
                    new Parameter() { Name = "parameter_vej_export_config", Value =
 JsonSerializer.Serialize(exportConfig) },
                    new Parameter() { Name = "parameter_step_1_vej_config", Value =
 JsonSerializer.Serialize(jobConfig) }
                }
            });
#pragma warning restore SageMaker1002
        return startExecutionResponse.PipelineExecutionArn;
    }
```

- Para API obtener más información, consulte StartPipelineExecutionla AWS SDK for .NET APIReferencia.

`UpdatePipeline`

En el siguiente ejemplo de código, se muestra cómo usar `UpdatePipeline`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
 already exists.
```

```
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });

            return createResponse.PipelineArn;
        }
    }
```

- Para API obtener más información, consulte UpdatePipelinela AWS SDK for .NET APIReferencia.

Escenarios

Introducción a las tareas y las canalizaciones geoespaciales

En el siguiente ejemplo de código, se muestra cómo:

- Configurar los recursos de una canalización
- Configurar una canalización que ejecuta un trabajo geoespacial
- Iniciar la ejecución de una canalización.
- Supervisar el estado de la ejecución.
- Ver el resultado de la canalización.
- Limpiar recursos.

Para obtener más información, consulte Crear y ejecutar SageMaker canalizaciones con AWS SDKs Community.aws.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crea una clase que abarque SageMaker las operaciones.

```
using System.Text.Json;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

namespace SageMakerActions;

/// <summary>
/// Wrapper class for Amazon SageMaker actions and logic.
/// </summary>
public class SageMakerWrapper
```

```
{
    private readonly IAmazonSageMaker _amazonSageMaker;
    public SageMakerWrapper(IAmazonSageMaker amazonSageMaker)
    {
        _amazonSageMaker = amazonSageMaker;
    }

    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
 already exists.
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
 string name, string description, string displayName)
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });

            return createResponse.PipelineArn;
        }
    }
```

```
    /// <summary>
    /// Run a pipeline with input and output file locations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
    /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
    /// <param name="outputLocationUrl">The output location in Amazon S3.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="executionRoleArn">The ARN of the role.</param>
    /// <returns>The ARN of the pipeline run.</returns>
    public async Task<string> ExecutePipeline(
        string queueUrl,
        string inputLocationUrl,
        string outputLocationUrl,
        string pipelineName,
        string executionRoleArn)
    {
        var inputConfig = new VectorEnrichmentJobInputConfig()
        {
            DataSourceConfig = new()
            {
                S3Data = new VectorEnrichmentJobS3Data()
                {
                    S3Uri = inputLocationUrl
                }
            },
            DocumentType = VectorEnrichmentJobDocumentType.CSV
        };

        var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = outputLocationUrl
            }
        };

        var jobConfig = new VectorEnrichmentJobConfig()
        {
            ReverseGeocodingConfig = new ReverseGeocodingConfig()
            {
                XAttributeName = "Longitude",
                YAttributeName = "Latitude"
```

```
            }
        };

#pragma warning disable SageMaker1002 // Property value does not match required
 pattern is allowed here to match the pipeline definition.
        var startExecutionResponse = await
 _amazonSageMaker.StartPipelineExecutionAsync(
            new StartPipelineExecutionRequest()
            {
                PipelineName = pipelineName,
                PipelineExecutionDisplayName = pipelineName + "-example-execution",
                PipelineParameters = new List<Parameter>()
                {
                    new Parameter() { Name = "parameter_execution_role", Value =
 executionRoleArn },
                    new Parameter() { Name = "parameter_queue_url", Value =
 queueUrl },
                    new Parameter() { Name = "parameter_vej_input_config", Value =
 JsonSerializer.Serialize(inputConfig) },
                    new Parameter() { Name = "parameter_vej_export_config", Value =
 JsonSerializer.Serialize(exportConfig) },
                    new Parameter() { Name = "parameter_step_1_vej_config", Value =
 JsonSerializer.Serialize(jobConfig) }
                }
            });
#pragma warning restore SageMaker1002
        return startExecutionResponse.PipelineExecutionArn;
    }

    /// <summary>
    /// Check the status of a run.
    /// </summary>
    /// <param name="pipelineExecutionArn">The ARN.</param>
    /// <returns>The status of the pipeline.</returns>
    public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
 pipelineExecutionArn)
    {
        var describeResponse = await
 _amazonSageMaker.DescribePipelineExecutionAsync(
            new DescribePipelineExecutionRequest()
            {
                PipelineExecutionArn = pipelineExecutionArn
            });
```

```
        return describeResponse.PipelineExecutionStatus;
    }

    /// <summary>
    /// Delete a SageMaker pipeline by name.
    /// </summary>
    /// <param name="pipelineName">The name of the pipeline to delete.</param>
    /// <returns>The ARN of the pipeline.</returns>
    public async Task<string> DeletePipelineByName(string pipelineName)
    {
        var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
            new DeletePipelineRequest()
            {
                PipelineName = pipelineName
            });

        return deleteResponse.PipelineArn;
    }
}
```

Crea una función que gestione las devoluciones de llamadas de la SageMaker canalización.

```
using System.Text.Json;
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

// Assembly attribute to enable the AWS Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SageMakerLambda;

/// <summary>
/// The AWS Lambda function handler for the Amazon SageMaker pipeline.
/// </summary>
public class SageMakerLambdaFunction
```

```
{
    /// <summary>
    /// Default constructor. This constructor is used by AWS Lambda to construct the
 instance. When invoked in a Lambda environment
    /// the AWS credentials will come from the AWS Identity and Access Management
 (IAM) role associated with the function. The AWS Region will be set to the
    /// Region that the Lambda function is running in.
    /// </summary>
    public SageMakerLambdaFunction()
    {
    }

    /// <summary>
    /// The AWS Lambda function handler that processes events from the SageMaker
 pipeline and starts a job or export.
    /// </summary>
    /// <param name="request">The custom SageMaker pipeline request object.</param>
    /// <param name="context">The Lambda context.</param>
    /// <returns>The dictionary of output parameters.</returns>
    public async Task<Dictionary<string, string>> FunctionHandler(PipelineRequest
 request, ILambdaContext context)
    {
        var geoSpatialClient = new AmazonSageMakerGeospatialClient();
        var sageMakerClient = new AmazonSageMakerClient();
        var responseDictionary = new Dictionary<string, string>();
        context.Logger.LogInformation("Function handler started with request: " +
 JsonSerializer.Serialize(request));
        if (request.Records != null && request.Records.Any())
        {
            context.Logger.LogInformation("Records found, this is a queue event.
 Processing the queue records.");
            foreach (var message in request.Records)
            {
                await ProcessMessageAsync(message, context, geoSpatialClient,
 sageMakerClient);
            }
        }
        else if (!string.IsNullOrEmpty(request.vej_export_config))
        {
            context.Logger.LogInformation("Export configuration found, this is an
 export. Start the Vector Enrichment Job (VEJ) export.");

            var outputConfig =
                JsonSerializer.Deserialize<ExportVectorEnrichmentJobOutputConfig>(
```

```
                    request.vej_export_config);

            var exportResponse = await
geoSpatialClient.ExportVectorEnrichmentJobAsync(
                new ExportVectorEnrichmentJobRequest()
                {
                    Arn = request.vej_arn,
                    ExecutionRoleArn = request.Role,
                    OutputConfig = outputConfig
                });
            context.Logger.LogInformation($"Export response:
{JsonSerializer.Serialize(exportResponse)}");
            responseDictionary = new Dictionary<string, string>
            {
                { "export_eoj_status", exportResponse.ExportStatus.ToString() },
                { "vej_arn", exportResponse.Arn }
            };
        }
        else if (!string.IsNullOrEmpty(request.vej_name))
        {
            context.Logger.LogInformation("Vector Enrichment Job name found,
starting the job.");
            var inputConfig =
                JsonSerializer.Deserialize<VectorEnrichmentJobInputConfig>(
                    request.vej_input_config);

            var jobConfig =
                JsonSerializer.Deserialize<VectorEnrichmentJobConfig>(
                    request.vej_config);

            var jobResponse = await geoSpatialClient.StartVectorEnrichmentJobAsync(
                new StartVectorEnrichmentJobRequest()
                {
                    ExecutionRoleArn = request.Role,
                    InputConfig = inputConfig,
                    Name = request.vej_name,
                    JobConfig = jobConfig

                });
            context.Logger.LogInformation("Job response: " +
JsonSerializer.Serialize(jobResponse));
            responseDictionary = new Dictionary<string, string>
            {
                { "vej_arn", jobResponse.Arn },
```

```
                { "statusCode", jobResponse.HttpStatusCode.ToString() }
            };
        }
        return responseDictionary;
    }

    /// <summary>
    /// Process a queue message and check the status of a SageMaker job.
    /// </summary>
    /// <param name="message">The queue message.</param>
    /// <param name="context">The Lambda context.</param>
    /// <param name="geoClient">The SageMaker GeoSpatial client.</param>
    /// <param name="sageMakerClient">The SageMaker client.</param>
    /// <returns>Async task.</returns>
    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context,
        AmazonSageMakerGeospatialClient geoClient, AmazonSageMakerClient
sageMakerClient)
    {
        context.Logger.LogInformation($"Processed message {message.Body}");

        // Get information about the SageMaker job.
        var payload = JsonSerializer.Deserialize<QueuePayload>(message.Body);
        context.Logger.LogInformation($"Payload token {payload!.token}");
        var token = payload.token;

        if (payload.arguments.ContainsKey("vej_arn"))
        {
            // Use the job ARN and the token to get the job status.
            var job_arn = payload.arguments["vej_arn"];
            context.Logger.LogInformation($"Token: {token}, arn {job_arn}");

            var jobInfo = geoClient.GetVectorEnrichmentJobAsync(
                new GetVectorEnrichmentJobRequest()
                {
                    Arn = job_arn
                });
            context.Logger.LogInformation("Job info: " +
JsonSerializer.Serialize(jobInfo));
            if (jobInfo.Result.Status == VectorEnrichmentJobStatus.COMPLETED)
            {
                context.Logger.LogInformation($"Status completed, resuming
pipeline...");
                await sageMakerClient.SendPipelineExecutionStepSuccessAsync(
```

```
                    new SendPipelineExecutionStepSuccessRequest()
                    {
                        CallbackToken = token,
                        OutputParameters = new List<OutputParameter>()
                        {
                            new OutputParameter()
                                { Name = "export_status", Value =
 jobInfo.Result.Status }
                        }
                    });
            }
            else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.FAILED)
            {
                context.Logger.LogInformation($"Status failed, stopping
 pipeline...");
                await sageMakerClient.SendPipelineExecutionStepFailureAsync(
                    new SendPipelineExecutionStepFailureRequest()
                    {
                        CallbackToken = token,
                        FailureReason = jobInfo.Result.ErrorDetails.ErrorMessage
                    });
            }
            else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.IN_PROGRESS)
            {
                // Put this message back in the queue to reprocess later.
                context.Logger.LogInformation(
                    $"Status still in progress, check back later.");
                throw new("Job still running.");
            }
        }
    }
}
```

Ejecutar un escenario interactivo en un símbolo del sistema.

```
public static class PipelineWorkflow
{
    public static IAmazonIdentityManagementService _iamClient = null!;
    public static SageMakerWrapper _sageMakerWrapper = null!;
    public static IAmazonSQS _sqsClient = null!;
    public static IAmazonS3 _s3Client = null!;
    public static IAmazonLambda _lambdaClient = null!;
```

```
    public static IConfiguration _configuration = null!;

    public static string lambdaFunctionName = "SageMakerExampleFunction";
    public static string sageMakerRoleName = "SageMakerExampleRole";
    public static string lambdaRoleName = "SageMakerExampleLambdaRole";

    private static string[] lambdaRolePolicies = null!;
    private static string[] sageMakerRolePolicies = null!;

    static async Task Main(string[] args)
    {
        var options = new AWSOptions() { Region = RegionEndpoint.USWest2 };
        // Set up dependency injection for the AWS service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIdentityManagementService>(options)
                    .AddAWSService<IAmazonEC2>(options)
                    .AddAWSService<IAmazonSageMaker>(options)
                    .AddAWSService<IAmazonSageMakerGeospatial>(options)
                    .AddAWSService<IAmazonSQS>(options)
                    .AddAWSService<IAmazonS3>(options)
                    .AddAWSService<IAmazonLambda>(options)
                    .AddTransient<SageMakerWrapper>()
        )
        .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        ServicesSetup(host);
        string queueUrl = "";
        string queueName = _configuration["queueName"];
        string bucketName = _configuration["bucketName"];
        var pipelineName = _configuration["pipelineName"];
```

```
        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(
                "Welcome to the Amazon SageMaker pipeline example scenario.");
            Console.WriteLine(
                "\nThis example workflow will guide you through setting up and
running an" +
                "\nAmazon SageMaker pipeline. The pipeline uses an AWS Lambda
function and an" +
                "\nAmazon SQS Queue. It runs a vector enrichment reverse geocode job
to" +
                "\nreverse geocode addresses in an input file and store the results
in an export file.");
            Console.WriteLine(new string('-', 80));

            Console.WriteLine(new string('-', 80));
            Console.WriteLine(
                "First, we will set up the roles, functions, and queue needed by the
SageMaker pipeline.");
            Console.WriteLine(new string('-', 80));

            var lambdaRoleArn = await CreateLambdaRole();
            var sageMakerRoleArn = await CreateSageMakerRole();
            var functionArn = await SetupLambda(lambdaRoleArn, true);
            queueUrl = await SetupQueue(queueName);
            await SetupBucket(bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Now we can create and run our pipeline.");
            Console.WriteLine(new string('-', 80));

            await SetupPipeline(sageMakerRoleArn, functionArn, pipelineName);
            var executionArn = await ExecutePipeline(queueUrl, sageMakerRoleArn,
pipelineName, bucketName);
            await WaitForPipelineExecution(executionArn);

            await GetOutputResults(bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("The pipeline has completed. To view the pipeline and
runs " +
                              "in SageMaker Studio, follow these instructions:" +
```

```
                                "\nhttps://docs.aws.amazon.com/sagemaker/latest/dg/
pipelines-studio.html");
            Console.WriteLine(new string('-', 80));

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await CleanupResources(true, queueUrl, pipelineName, bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("SageMaker pipeline scenario is complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
 {ex.Message}");
            await CleanupResources(true, queueUrl, pipelineName, bucketName);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _sageMakerWrapper = host.Services.GetRequiredService<SageMakerWrapper>();
        _iamClient =
 host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _sqsClient = host.Services.GetRequiredService<IAmazonSQS>();
        _s3Client = host.Services.GetRequiredService<IAmazonS3>();
        _lambdaClient = host.Services.GetRequiredService<IAmazonLambda>();
    }

    /// <summary>
    /// Set up AWS Lambda, either by updating an existing function or creating a new
 function.
    /// </summary>
    /// <param name="roleArn">The role Amazon Resource Name (ARN) to use for the
 Lambda function.</param>
```

```
    /// <param name="askUser">True to ask the user before updating.</param>
    /// <returns>The ARN of the function.</returns>
    public static async Task<string> SetupLambda(string roleArn, bool askUser)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Setting up the Lambda function for the pipeline.");
        var handlerName =
"SageMakerLambda::SageMakerLambda.SageMakerLambdaFunction::FunctionHandler";
        var functionArn = "";
        try
        {
            var functionInfo = await _lambdaClient.GetFunctionAsync(new
GetFunctionRequest()
            {
                FunctionName = lambdaFunctionName
            });

            var updateFunction = true;
            if (askUser)
            {
                updateFunction = GetYesNoResponse(
                    $"\tThe Lambda function {lambdaFunctionName} already exists, do
you want to update it?");
            }

            if (updateFunction)
            {
                // Update the Lambda function.
                using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
                await _lambdaClient.UpdateFunctionCodeAsync(
                    new UpdateFunctionCodeRequest()
                    {
                        FunctionName = lambdaFunctionName,
                        ZipFile = zipMemoryStream,
                    });
            }

            functionArn = functionInfo.Configuration.FunctionArn;
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"\tThe Lambda function {lambdaFunctionName} was not
found, creating the new function.");
```

```
            // Create the function if it does not already exist.
            using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
            var createResult = await _lambdaClient.CreateFunctionAsync(
                new CreateFunctionRequest()
                {
                    FunctionName = lambdaFunctionName,
                    Runtime = Runtime.Dotnet6,
                    Description = "SageMaker example function.",
                    Code = new FunctionCode()
                    {
                        ZipFile = zipMemoryStream
                    },
                    Handler = handlerName,
                    Role = roleArn,
                    Timeout = 30
                });

            functionArn = createResult.FunctionArn;
        }

        Console.WriteLine($"\tLambda ready with ARN {functionArn}.");
        Console.WriteLine(new string('-', 80));
        return functionArn;
    }

    /// <summary>
    /// Create a role to be used by AWS Lambda. Does not create the role if it
already exists.
    /// </summary>
    /// <returns>The role ARN.</returns>
    public static async Task<string> CreateLambdaRole()
    {
        Console.WriteLine(new string('-', 80));

        lambdaRolePolicies = new string[]{
            "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
            "arn:aws:iam::aws:policy/AmazonSQSFullAccess",
            "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerGeospatialFullAccess",
            "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy",
```

```
            "arn:aws:iam::aws:policy/service-role/" +
 "AWSLambdaSQSQueueExecutionRole"
        };

        var roleArn = await GetRoleArnIfExists(lambdaRoleName);
        if (!string.IsNullOrEmpty(roleArn))
        {
            return roleArn;
        }

        Console.WriteLine("\tCreating a role to for AWS Lambda to use.");

        var assumeRolePolicy = "{" +
                               "\"Version\": \"2012-10-17\"," +
                               "\"Statement\": [{" +
                                    "\"Effect\": \"Allow\"," +
                                    "\"Principal\": {" +
                                        $"\"Service\": [" +
                                            "\"sagemaker.amazonaws.com\"," +
                                            "\"sagemaker-geospatial.amazonaws.com
\"," +
                                            "\"lambda.amazonaws.com\"," +
                                            "\"s3.amazonaws.com\"" +
                                        "]" +
                                    "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                               "}]" +
                           "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = lambdaRoleName
            });
        foreach (var policy in lambdaRolePolicies)
        {
            await _iamClient.AttachRolePolicyAsync(
                new AttachRolePolicyRequest()
                {
                    PolicyArn = policy,
                    RoleName = lambdaRoleName
                });
```

```
        }

        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        Console.WriteLine($"\tRole ready with ARN {roleResult.Role.Arn}.");
        Console.WriteLine(new string('-', 80));

        return roleResult.Role.Arn;
    }


    /// <summary>
    /// Create a role to be used by SageMaker.
    /// </summary>
    /// <returns>The role Amazon Resource Name (ARN).</returns>
    public static async Task<string> CreateSageMakerRole()
    {
        Console.WriteLine(new string('-', 80));

        sageMakerRolePolicies = new string[]{
            "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
            "arn:aws:iam::aws:policy/AmazonSageMakerGeospatialFullAccess",
        };

        var roleArn = await GetRoleArnIfExists(sageMakerRoleName);
        if (!string.IsNullOrEmpty(roleArn))
        {
            return roleArn;
        }

        Console.WriteLine("\tCreating a role to use with SageMaker.");

        var assumeRolePolicy = "{" +
                                        "\"Version\": \"2012-10-17\"," +
                                        "\"Statement\": [{" +
                                            "\"Effect\": \"Allow\"," +
                                            "\"Principal\": {" +
                                                $"\"Service\": [" +
                                                    "\"sagemaker.amazonaws.com\"," +
                                                    "\"sagemaker-
geospatial.amazonaws.com\"," +
                                                    "\"lambda.amazonaws.com\"," +
                                                    "\"s3.amazonaws.com\"" +
                                                "]" +
```

```
                                        "}," +
                                        "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = sageMakerRoleName
            });

        foreach (var policy in sageMakerRolePolicies)
        {
            await _iamClient.AttachRolePolicyAsync(
                new AttachRolePolicyRequest()
                {
                    PolicyArn = policy,
                    RoleName = sageMakerRoleName
                });
        }

        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        Console.WriteLine($"\tRole ready with ARN {roleResult.Role.Arn}.");
        Console.WriteLine(new string('-', 80));
        return roleResult.Role.Arn;
    }

    /// <summary>
    /// Set up the SQS queue to use with the pipeline.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <returns>The URL for the queue.</returns>
    public static async Task<string> SetupQueue(string queueName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up queue {queueName}.");

        try
        {
            var queueInfo = await _sqsClient.GetQueueUrlAsync(new
GetQueueUrlRequest()
```

```
            { QueueName = queueName });
            return queueInfo.QueueUrl;
        }
        catch (QueueDoesNotExistException)
        {
            var attrs = new Dictionary<string, string>
            {
                {
                    QueueAttributeName.DelaySeconds,
                    "5"
                },
                {
                    QueueAttributeName.ReceiveMessageWaitTimeSeconds,
                    "5"
                },
                {
                    QueueAttributeName.VisibilityTimeout,
                    "300"
                },
            };

            var request = new CreateQueueRequest
            {
                Attributes = attrs,
                QueueName = queueName,
            };

            var response = await _sqsClient.CreateQueueAsync(request);
            Thread.Sleep(10000);
            await ConnectLambda(response.QueueUrl);
            Console.WriteLine($"\tQueue ready with Url {response.QueueUrl}.");
            Console.WriteLine(new string('-', 80));
            return response.QueueUrl;
        }
    }

    /// <summary>
    /// Connect the queue to the Lambda function as an event source.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue.</param>
    /// <returns>Async task.</returns>
    public static async Task ConnectLambda(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"Connecting the Lambda function and queue for the
pipeline.");

        var queueAttributes = await _sqsClient.GetQueueAttributesAsync(
            new GetQueueAttributesRequest() { QueueUrl = queueUrl, AttributeNames =
new List<string>() { "All" } });
        var queueArn = queueAttributes.QueueARN;

        var eventSource = await _lambdaClient.ListEventSourceMappingsAsync(
             new ListEventSourceMappingsRequest()
             {
                 FunctionName = lambdaFunctionName
             });

        if (!eventSource.EventSourceMappings.Any())
        {
            // Only add the event source mapping if it does not already exist.
            await _lambdaClient.CreateEventSourceMappingAsync(
                new CreateEventSourceMappingRequest()
                {
                    EventSourceArn = queueArn,
                    FunctionName = lambdaFunctionName,
                    Enabled = true
                });
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the bucket to use for pipeline input and output.
    /// </summary>
    /// <param name="bucketName">The name for the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task SetupBucket(string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up bucket {bucketName}.");

        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            bucketName);

        if (!bucketExists)
```

```
        {
            await _s3Client.PutBucketAsync(new PutBucketRequest()
            {
                BucketName = bucketName,
                BucketRegion = S3Region.USWest2
            });

            Thread.Sleep(5000);

            await _s3Client.PutObjectAsync(new PutObjectRequest()
            {
                BucketName = bucketName,
                Key = "samplefiles/latlongtest.csv",
                FilePath = "latlongtest.csv"
            });
        }

        Console.WriteLine($"\tBucket {bucketName} ready.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Display some results from the output directory.
    /// </summary>
    /// <param name="bucketName">The name for the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<string> GetOutputResults(string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Getting output results {bucketName}.");
        string outputKey = "";
        Thread.Sleep(15000);
        var outputFiles = await _s3Client.ListObjectsAsync(
              new ListObjectsRequest()
              {
                  BucketName = bucketName,
                  Prefix = "outputfiles/"
              });

        if (outputFiles.S3Objects.Any())
        {
            var sampleOutput = outputFiles.S3Objects.OrderBy(s =>
s.LastModified).Last();
            Console.WriteLine($"\tOutput file: {sampleOutput.Key}");
```

```
            var outputSampleResponse = await _s3Client.GetObjectAsync(
                new GetObjectRequest()
                {
                    BucketName = bucketName,
                    Key = sampleOutput.Key
                });
            outputKey = sampleOutput.Key;
            StreamReader reader = new
 StreamReader(outputSampleResponse.ResponseStream);
            await reader.ReadLineAsync();
            Console.WriteLine("\tOutput file contents: \n");
            for (int i = 0; i < 10; i++)
            {
                if (!reader.EndOfStream)
                {
                    Console.WriteLine("\t" + await reader.ReadLineAsync());
                }
            }
        }

        Console.WriteLine(new string('-', 80));
        return outputKey;
    }

    /// <summary>
    /// Create a pipeline from the example pipeline JSON
    /// that includes the Lambda, callback, processing, and export jobs.
    /// </summary>
    /// <param name="roleArn">The ARN of the role for the pipeline.</param>
    /// <param name="functionArn">The ARN of the Lambda function for the pipeline.</
param>
    /// <param name="pipelineName">The name for the pipeline.</param>
    /// <returns>The ARN of the pipeline.</returns>
    public static async Task<string> SetupPipeline(string roleArn, string
 functionArn, string pipelineName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up the pipeline.");

        var pipelineJson = await File.ReadAllTextAsync("GeoSpatialPipeline.json");

        // Add the correct function ARN instead of the placeholder.
        pipelineJson = pipelineJson.Replace("*FUNCTION_ARN*", functionArn);
```

```
        var pipelineArn = await _sageMakerWrapper.SetupPipeline(pipelineJson,
roleArn, pipelineName,
            "sdk example pipeline", pipelineName);

        Console.WriteLine($"\tPipeline set up with ARN {pipelineArn}.");
        Console.WriteLine(new string('-', 80));

        return pipelineArn;
    }

    /// <summary>
    /// Start a pipeline run with job configurations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue used in the pipeline.</param>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>The pipeline run ARN.</returns>
    public static async Task<string> ExecutePipeline(
        string queueUrl,
        string roleArn,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Starting pipeline execution.");

        var input = $"s3://{bucketName}/samplefiles/latlongtest.csv";
        var output = $"s3://{bucketName}/outputfiles/";

        var executionARN =
            await _sageMakerWrapper.ExecutePipeline(queueUrl, input, output,
                pipelineName, roleArn);

        Console.WriteLine($"\tRun started with ARN {executionARN}.");
        Console.WriteLine(new string('-', 80));

        return executionARN;
    }

    /// <summary>
    /// Wait for a pipeline run to complete.
    /// </summary>
    /// <param name="executionArn">The pipeline run ARN.</param>
```

```
    /// <returns>Async task.</returns>
    public static async Task WaitForPipelineExecution(string executionArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Waiting for pipeline to finish.");

        PipelineExecutionStatus status;
        do
        {
            status = await
 _sageMakerWrapper.CheckPipelineExecutionStatus(executionArn);
            Thread.Sleep(30000);
            Console.WriteLine($"\tStatus is {status}.");
        } while (status == PipelineExecutionStatus.Executing);

        Console.WriteLine($"\tPipeline finished with status {status}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="askUser">True to ask the user for cleanup.</param>
    /// <param name="queueUrl">The URL of the queue to clean up.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> CleanupResources(
        bool askUser,
        string queueUrl,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (!askUser || GetYesNoResponse($"\tDelete pipeline {pipelineName}? (y/
n)"))
        {
            Console.WriteLine($"\tDeleting pipeline.");
            // Delete the pipeline.
            await _sageMakerWrapper.DeletePipelineByName(pipelineName);
        }
```

```
        if (!string.IsNullOrEmpty(queueUrl) && (!askUser ||
 GetYesNoResponse($"\tDelete queue {queueUrl}? (y/n)")))
        {
            Console.WriteLine($"\tDeleting queue.");
            // Delete the queue.
            await _sqsClient.DeleteQueueAsync(new DeleteQueueRequest(queueUrl));
        }

        if (!askUser || GetYesNoResponse($"\tDelete Amazon S3 bucket {bucketName}?
 (y/n)"))
        {
            Console.WriteLine($"\tDeleting bucket.");
            // Delete all objects in the bucket.
            var deleteList = await _s3Client.ListObjectsV2Async(new
 ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            if (deleteList.KeyCount > 0)
            {
                await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.S3Objects
                        .Select(o => new KeyVersion { Key = o.Key }).ToList()
                });
            }

            // Now delete the bucket.
            await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
            {
                BucketName = bucketName
            });
        }

        if (!askUser || GetYesNoResponse($"\tDelete lambda {lambdaFunctionName}? (y/
n)"))
        {
            Console.WriteLine($"\tDeleting lambda function.");

            await _lambdaClient.DeleteFunctionAsync(new DeleteFunctionRequest()
            {
                FunctionName = lambdaFunctionName
            });
```

```
        }

        if (!askUser || GetYesNoResponse($"\tDelete role {lambdaRoleName}? (y/n)"))
        {
            Console.WriteLine($"\tDetaching policies and deleting role.");

            foreach (var policy in lambdaRolePolicies)
            {
                await _iamClient!.DetachRolePolicyAsync(new
 DetachRolePolicyRequest()
                {
                    RoleName = lambdaRoleName,
                    PolicyArn = policy
                });
            }

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = lambdaRoleName
            });
        }

        if (!askUser || GetYesNoResponse($"\tDelete role {sageMakerRoleName}? (y/
n)"))
        {
            Console.WriteLine($"\tDetaching policies and deleting role.");

            foreach (var policy in sageMakerRolePolicies)
            {
                await _iamClient!.DetachRolePolicyAsync(new
 DetachRolePolicyRequest()
                {
                    RoleName = sageMakerRoleName,
                    PolicyArn = policy
                });
            }

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = sageMakerRoleName
            });
        }

        Console.WriteLine(new string('-', 80));
```

```
        return true;
    }

    /// <summary>
    /// Helper method to get a role's ARN if it already exists.
    /// </summary>
    /// <param name="roleName">The name of the AWS Identity and Access Management
 (IAM) Role to look for.</param>
    /// <returns>The role ARN if it exists, otherwise an empty string.</returns>
    private static async Task<string> GetRoleArnIfExists(string roleName)
    {
        Console.WriteLine($"Checking for role named {roleName}.");

        try
        {
            var existingRole = await _iamClient.GetRoleAsync(new GetRoleRequest()
            {
                RoleName = lambdaRoleName
            });
            return existingRole.Role.Arn;
        }
        catch (NoSuchEntityException)
        {
            return string.Empty;
        }
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreatePipeline
  - DeletePipeline
  - DescribePipelineExecution
  - StartPipelineExecution
  - UpdatePipeline

Ejemplos de Secrets Manager usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Secrets Manager.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones

Acciones

`GetSecretValue`

En el siguiente ejemplo de código, se muestra cómo usar `GetSecretValue`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.SecretsManager;
using Amazon.SecretsManager.Model;

/// <summary>
/// This example uses the Amazon Web Service Secrets Manager to retrieve
/// the secret value for the provided secret name.
/// </summary>
public class GetSecretValue
{
    /// <summary>
    /// The main method initializes the necessary values and then calls
    /// the GetSecretAsync and DecodeString methods to get the decoded
    /// secret value for the secret named in secretName.
    /// </summary>
    public static async Task Main()
    {
        string secretName = "<<{{MySecretName}}>>";
        string secret;

        IAmazonSecretsManager client = new AmazonSecretsManagerClient();

        var response = await GetSecretAsync(client, secretName);

        if (response is not null)
        {
            secret = DecodeString(response);

            if (!string.IsNullOrEmpty(secret))
            {
                Console.WriteLine($"The decoded secret value is: {secret}.");
            }
            else
            {
                Console.WriteLine("No secret value was returned.");
            }
        }
    }

    /// <summary>
    /// Retrieves the secret value given the name of the secret to
```

```
        /// retrieve.
        /// </summary>
        /// <param name="client">The client object used to retrieve the secret
        /// value for the given secret name.</param>
        /// <param name="secretName">The name of the secret value to retrieve.</
param>
        /// <returns>The GetSecretValueReponse object returned by
        /// GetSecretValueAsync.</returns>
        public static async Task<GetSecretValueResponse> GetSecretAsync(
            IAmazonSecretsManager client,
            string secretName)
        {
            GetSecretValueRequest request = new GetSecretValueRequest()
            {
                SecretId = secretName,
                VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT
 if unspecified.
            };

            GetSecretValueResponse response = null;

            // For the sake of simplicity, this example handles only the most
            // general SecretsManager exception.
            try
            {
                response = await client.GetSecretValueAsync(request);
            }
            catch (AmazonSecretsManagerException e)
            {
                Console.WriteLine($"Error: {e.Message}");
            }

            return response;
        }

        /// <summary>
        /// Decodes the secret returned by the call to GetSecretValueAsync and
        /// returns it to the calling program.
        /// </summary>
        /// <param name="response">A GetSecretValueResponse object containing
        /// the requested secret value returned by GetSecretValueAsync.</param>
        /// <returns>A string representing the decoded secret value.</returns>
        public static string DecodeString(GetSecretValueResponse response)
        {
```

```
            // Decrypts secret using the associated AWS Key Management Service
            // Customer Master Key (CMK.) Depending on whether the secret is a
            // string or binary value, one of these fields will be populated.
            if (response.SecretString is not null)
            {
                var secret = response.SecretString;
                return secret;
            }
            else if (response.SecretBinary is not null)
            {
                var memoryStream = response.SecretBinary;
                StreamReader reader = new StreamReader(memoryStream);
                string decodedBinarySecret =
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(reader.ReadToEnd()));
                return decodedBinarySecret;
            }
            else
            {
                return string.Empty;
            }
        }
    }
```

- Para API obtener más información, consulte GetSecretValuela AWS SDK for .NET APIReferencia.

SESEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonSES.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`CreateTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `CreateTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create an email template.
   /// </summary>
   /// <param name="name">Name of the template.</param>
   /// <param name="subject">Email subject.</param>
   /// <param name="text">Email body text.</param>
   /// <param name="html">Email HTML body text.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> CreateEmailTemplateAsync(string name, string subject,
string text,
       string html)
   {
       var success = false;
       try
       {
           var response = await _amazonSimpleEmailService.CreateTemplateAsync(
               new CreateTemplateRequest
               {
                   Template = new Template
                   {
                       TemplateName = name,
                       SubjectPart = subject,
```

```
                        TextPart = text,
                        HtmlPart = html
                    }
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("CreateEmailTemplateAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- Para API obtener más información, consulte CreateTemplatela AWS SDK for .NET APIReferencia.

`DeleteIdentity`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteIdentity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an email identity.
    /// </summary>
    /// <param name="identityEmail">The identity email to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteIdentityAsync(string identityEmail)
    {
        var success = false;
```

```
        try
        {
            var response = await _amazonSimpleEmailService.DeleteIdentityAsync(
                new DeleteIdentityRequest
                {
                    Identity = identityEmail
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteIdentityAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- Para API obtener más información, consulte DeleteIdentityla AWS SDK for .NET APIReferencia.

`DeleteTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete an email template.
    /// </summary>
    /// <param name="templateName">Name of the template.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.DeleteTemplateAsync(
                new DeleteTemplateRequest
                {
                    TemplateName = templateName
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteEmailTemplateAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- Para API obtener más información, consulte DeleteTemplatela AWS SDK for .NET APIReferencia.

`GetIdentityVerificationAttributes`

En el siguiente ejemplo de código, se muestra cómo usar `GetIdentityVerificationAttributes`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
```

```
    /// Get identity verification status for an email.
    /// </summary>
    /// <returns>The verification status of the email.</returns>
    public async Task<VerificationStatus> GetIdentityStatusAsync(string email)
    {
        var result = VerificationStatus.TemporaryFailure;
        try
        {
            var response =
                await
_amazonSimpleEmailService.GetIdentityVerificationAttributesAsync(
                    new GetIdentityVerificationAttributesRequest
                    {
                        Identities = new List<string> { email }
                    });

            if (response.VerificationAttributes.ContainsKey(email))
                result = response.VerificationAttributes[email].VerificationStatus;
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetIdentityStatusAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- Para API obtener más información, consulte GetIdentityVerificationAttributesla AWS SDK for .NET APIReferencia.

`GetSendQuota`

En el siguiente ejemplo de código, se muestra cómo usar `GetSendQuota`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information on the current account's send quota.
    /// </summary>
    /// <returns>The send quota response data.</returns>
    public async Task<GetSendQuotaResponse> GetSendQuotaAsync()
    {
        var result = new GetSendQuotaResponse();
        try
        {
            var response = await _amazonSimpleEmailService.GetSendQuotaAsync(
                new GetSendQuotaRequest());
            result = response;
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetSendQuotaAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- Para API obtener más información, consulte GetSendQuotala AWS SDK for .NET APIReferencia.

`ListIdentities`

En el siguiente ejemplo de código, se muestra cómo usar `ListIdentities`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the identities of a specified type for the current account.
    /// </summary>
    /// <param name="identityType">IdentityType to list.</param>
    /// <returns>The list of identities.</returns>
    public async Task<List<string>> ListIdentitiesAsync(IdentityType identityType)
    {
        var result = new List<string>();
        try
        {
            var response = await _amazonSimpleEmailService.ListIdentitiesAsync(
                new ListIdentitiesRequest
                {
                    IdentityType = identityType
                });
            result = response.Identities;
        }
        catch (Exception ex)
        {
            Console.WriteLine("ListIdentitiesAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- Para API obtener más información, consulte ListIdentitiesla AWS SDK for .NET APIReferencia.

`ListTemplates`

En el siguiente ejemplo de código, se muestra cómo usar `ListTemplates`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List email templates for the current account.
    /// </summary>
    /// <returns>A list of template metadata.</returns>
    public async Task<List<TemplateMetadata>> ListEmailTemplatesAsync()
    {
        var result = new List<TemplateMetadata>();
        try
        {
            var response = await _amazonSimpleEmailService.ListTemplatesAsync(
                new ListTemplatesRequest());
            result = response.TemplatesMetadata;
        }
        catch (Exception ex)
        {
            Console.WriteLine("ListEmailTemplatesAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- Para API obtener más información, consulte ListTemplatesla AWS SDK for .NET APIReferencia.

`SendEmail`

En el siguiente ejemplo de código, se muestra cómo usar `SendEmail`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
///  Send an email by using Amazon SES.
/// </summary>
/// <param name="toAddresses">List of recipients.</param>
/// <param name="ccAddresses">List of cc recipients.</param>
/// <param name="bccAddresses">List of bcc recipients.</param>
/// <param name="bodyHtml">Body of the email in HTML.</param>
/// <param name="bodyText">Body of the email in plain text.</param>
/// <param name="subject">Subject line of the email.</param>
/// <param name="senderAddress">From address.</param>
/// <returns>The messageId of the email.</returns>
public async Task<string> SendEmailAsync(List<string> toAddresses,
    List<string> ccAddresses, List<string> bccAddresses,
    string bodyHtml, string bodyText, string subject, string senderAddress)
{
    var messageId = "";
    try
    {
        var response = await _amazonSimpleEmailService.SendEmailAsync(
            new SendEmailRequest
            {
                Destination = new Destination
                {
                    BccAddresses = bccAddresses,
                    CcAddresses = ccAddresses,
                    ToAddresses = toAddresses
                },
                Message = new Message
                {
                    Body = new Body
```

```
                        {
                            Html = new Content
                            {
                                Charset = "UTF-8",
                                Data = bodyHtml
                            },
                            Text = new Content
                            {
                                Charset = "UTF-8",
                                Data = bodyText
                            }
                        },
                        Subject = new Content
                        {
                            Charset = "UTF-8",
                            Data = subject
                        }
                    },
                    Source = senderAddress
                });
            messageId = response.MessageId;
        }
        catch (Exception ex)
        {
            Console.WriteLine("SendEmailAsync failed with exception: " +
ex.Message);
        }

        return messageId;
    }
```

- Para API obtener más información, consulte SendEmailla AWS SDK for .NET APIReferencia.

`SendTemplatedEmail`

En el siguiente ejemplo de código, se muestra cómo usar `SendTemplatedEmail`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Send an email using a template.
   /// </summary>
   /// <param name="sender">Address of the sender.</param>
   /// <param name="recipients">Addresses of the recipients.</param>
   /// <param name="templateName">Name of the email template.</param>
   /// <param name="templateDataObject">Data for the email template.</param>
   /// <returns>The messageId of the email.</returns>
   public async Task<string> SendTemplateEmailAsync(string sender, List<string>
recipients,
       string templateName, object templateDataObject)
   {
       var messageId = "";
       try
       {
           // Template data should be serialized JSON from either a class or a
dynamic object.
           var templateData = JsonSerializer.Serialize(templateDataObject);

           var response = await _amazonSimpleEmailService.SendTemplatedEmailAsync(
               new SendTemplatedEmailRequest
               {
                   Source = sender,
                   Destination = new Destination
                   {
                       ToAddresses = recipients
                   },
                   Template = templateName,
                   TemplateData = templateData
               });
           messageId = response.MessageId;
       }
       catch (Exception ex)
```

```
        {
            Console.WriteLine("SendTemplateEmailAsync failed with exception: " +
ex.Message);
        }

        return messageId;
    }
```

- Para API obtener más información, consulte SendTemplatedEmailla AWS SDK for .NET APIReferencia.

`VerifyEmailIdentity`

En el siguiente ejemplo de código, se muestra cómo usar `VerifyEmailIdentity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Starts verification of an email identity. This request sends an email
    /// from Amazon SES to the specified email address. To complete
    /// verification, follow the instructions in the email.
    /// </summary>
    /// <param name="recipientEmailAddress">Email address to verify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyEmailIdentityAsync(string recipientEmailAddress)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.VerifyEmailIdentityAsync(
                new VerifyEmailIdentityRequest
                {
```

```
                    EmailAddress = recipientEmailAddress
                });

            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("VerifyEmailIdentityAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- Para API obtener más información, consulte VerifyEmailIdentityla AWS SDK for .NET APIReferencia.

Escenarios

Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una tabla de Amazon DynamoDB y utilice Amazon Simple Email Service (SESAmazon) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizar Amazon DynamoDB. NETAPIpara crear una aplicación web dinámica que realice un seguimiento de los datos de trabajo de DynamoDB.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. GitHub

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Crear un rastreador de elementos de trabajo de Aurora Serverless

El siguiente ejemplo de código muestra cómo crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos Amazon Aurora Serverless y utilice Amazon Simple Email Service (AmazonSES) para enviar informes.

AWS SDK for .NET

Muestra cómo utilizarla AWS SDK for .NET para crear una aplicación web que haga un seguimiento de los elementos de trabajo de una base de datos de Amazon Aurora y envíe informes por correo electrónico mediante Amazon Simple Email Service (AmazonSES). En este ejemplo, se utiliza una interfaz creada con React.js para interactuar con unRESTful. NETbackend.

- Integre una aplicación web de React con AWS los servicios.
- Muestre, agregue, actualice y elimine elementos en una tabla de Aurora.
- Envía un informe por correo electrónico de los artículos de trabajo filtrados a través de AmazonSES.
- Implemente y gestione recursos de ejemplo con el AWS CloudFormation script incluido.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Detectar objetos en imágenes

El siguiente ejemplo de código muestra cómo crear una aplicación que utilice Amazon Rekognition para detectar objetos por categoría en las imágenes.

AWS SDK for .NET

Muestra cómo usar Amazon Rekognition. NETAPIpara crear una aplicación que utilice Amazon Rekognition para identificar objetos por categoría en imágenes ubicadas en un bucket de Amazon

Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (AmazonSES).

Para obtener el código fuente completo y las instrucciones sobre cómo configurarla y ejecutarla, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES

Ejemplos de Amazon SES API v2 que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Amazon SES API v2.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Escenarios

Acciones

`CreateContact`

En el siguiente ejemplo de código, se muestra cómo usar `CreateContact`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Creates a contact and adds it to the specified contact list.
   /// </summary>
   /// <param name="emailAddress">The email address of the contact.</param>
   /// <param name="contactListName">The name of the contact list.</param>
   /// <returns>The response from the CreateContact operation.</returns>
   public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
   {
       var request = new CreateContactRequest
       {
           EmailAddress = emailAddress,
           ContactListName = contactListName
       };

       try
       {
           var response = await _sesClient.CreateContactAsync(request);
           return response.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (AlreadyExistsException ex)
       {
           Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
           Console.WriteLine(ex.Message);
           return true;
       }
       catch (NotFoundException ex)
       {
           Console.WriteLine($"The contact list {contactListName} does not
exist.");
           Console.WriteLine(ex.Message);
       }
       catch (TooManyRequestsException ex)
```

```
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }
```

- Para API obtener más información, consulte CreateContactla AWS SDK for .NET APIReferencia.

`CreateContactList`

En el siguiente ejemplo de código, se muestra cómo usar `CreateContactList`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
        {
            ContactListName = contactListName
        };
```

```
        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
        }
        return false;
    }
```

- Para API obtener más información, consulte CreateContactListla AWS SDK for .NET APIReferencia.

`CreateEmailIdentity`

En el siguiente ejemplo de código, se muestra cómo usar `CreateEmailIdentity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Creates an email identity (email address or domain) and starts the
verification process.
   /// </summary>
   /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
   /// <returns>The response from the CreateEmailIdentity operation.</returns>
   public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
   {
       var request = new CreateEmailIdentityRequest
       {
           EmailIdentity = emailIdentity
       };

       try
       {
           var response = await _sesClient.CreateEmailIdentityAsync(request);
           return response;
       }
       catch (AlreadyExistsException ex)
       {
           Console.WriteLine($"Email identity {emailIdentity} already exists.");
           Console.WriteLine(ex.Message);
           throw;
       }
       catch (ConcurrentModificationException ex)
       {
           Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
           Console.WriteLine(ex.Message);
           throw;
       }
       catch (LimitExceededException ex)
```

```
        {
            Console.WriteLine("The limit for email identities has been exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
            throw;
        }
    }
```

- Para API obtener más información, consulte CreateEmailIdentityla AWS SDK for .NET APIReferencia.

`CreateEmailTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `CreateEmailTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email template:
 {ex.Message}");
        }

        return false;
    }
```

- Para API obtener más información, consulte CreateEmailTemplatela AWS SDK for .NET APIReferencia.

`DeleteContactList`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteContactList`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteContactListAsync(string contactListName)
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
```

```
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
        }

        return false;
    }
```

- Para API obtener más información, consulte DeleteContactListla AWS SDK for .NET APIReferencia.

`DeleteEmailIdentity`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteEmailIdentity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
        }

        return false;
    }
```

- Para API obtener más información, consulte DeleteEmailIdentityla AWS SDK for .NET APIReferencia.

`DeleteEmailTemplate`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteEmailTemplate`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
```

```
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
        }

        return false;
    }
```

- Para API obtener más información, consulte DeleteEmailTemplatela AWS SDK for .NET APIReferencia.

`ListContacts`

En el siguiente ejemplo de código, se muestra cómo usar `ListContacts`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
```

```
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
 {ex.Message}");
        }

        return new List<Contact>();
    }
```

- Para API obtener más información, consulte ListContactsla AWS SDK for .NET APIReferencia.

`SendEmail`

En el siguiente ejemplo de código, se muestra cómo usar `SendEmail`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email from.</
param>
    /// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
    /// <param name="subject">The subject of the email.</param>
    /// <param name="htmlContent">The HTML content of the email.</param>
    /// <param name="textContent">The text content of the email.</param>
    /// <param name="templateName">The name of the email template to use
 (optional).</param>
    /// <param name="templateData">The data to replace placeholders in the email
 template (optional).</param>
    /// <param name="contactListName">The name of the contact list for unsubscribe
 functionality (optional).</param>
    /// <returns>The MessageId response from the SendEmail operation.</returns>
    public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
 toEmailAddresses, string? subject,
        string? htmlContent, string? textContent, string? templateName = null,
 string? templateData = null, string? contactListName = null)
    {
        var request = new SendEmailRequest
        {
            FromEmailAddress = fromEmailAddress
        };

        if (toEmailAddresses.Any())
        {
            request.Destination = new Destination { ToAddresses =
 toEmailAddresses };
        }

        if (!string.IsNullOrEmpty(templateName))
```

```
        {
            request.Content = new EmailContent()
            {
                Template = new Template
                {
                    TemplateName = templateName,
                    TemplateData = templateData
                }
            };
        }
        else
        {
            request.Content = new EmailContent
            {
                Simple = new Message
                {
                    Subject = new Content { Data = subject },
                    Body = new Body
                    {
                        Html = new Content { Data = htmlContent },
                        Text = new Content { Data = textContent }
                    }
                }
            };
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
```

```
        }
        catch (MailFromDomainNotVerifiedException ex)
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
        }

        return string.Empty;
    }
```

- Para API obtener más información, consulte SendEmailla AWS SDK for .NET APIReferencia.

Escenarios

flujo de trabajo del boletín

El siguiente ejemplo de código muestra cómo ejecutar el flujo de trabajo del boletín de Amazon SES API v2.

AWS SDK for .NET

> **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 el flujo de trabajo.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace Sesv2Scenario;

public static class NewsletterWorkflow
{
    /*
      This workflow demonstrates how to use the Amazon Simple Email Service (SES) v2
 to send a coupon newsletter to a list of subscribers.
      The workflow performs the following tasks:

      1. Prepare the application:
         - Create a verified email identity for sending and replying to emails.
         - Create a contact list to store the subscribers' email addresses.
         - Create an email template for the coupon newsletter.

      2. Gather subscriber email addresses:
         - Prompt the user for a base email address.
         - Create 3 variants of the email address using subaddress extensions (e.g.,
 user+ses-weekly-newsletter-1@example.com).
         - Add each variant as a contact to the contact list.
         - Send a welcome email to each new contact.

      3. Send the coupon newsletter:
         - Retrieve the list of contacts from the contact list.
```

```
         - Send the coupon newsletter using the email template to each contact.

     4. Monitor and review:
        - Provide instructions for the user to review the sending activity and
 metrics in the AWS console.

     5. Clean up resources:
        - Delete the contact list (which also deletes all contacts within it).
        - Delete the email template.
        - Optionally delete the verified email identity.

   */

   public static SESv2Wrapper _sesv2Wrapper;
   public static string? _baseEmailAddress = null;
   public static string? _verifiedEmail = null;
   private static string _contactListName = "weekly-coupons-newsletter";
   private static string _templateName = "weekly-coupons";
   private static string _subject = "Weekly Coupons Newsletter";
   private static string _htmlContentFile = "coupon-newsletter.html";
   private static string _textContentFile = "coupon-newsletter.txt";
   private static string _htmlWelcomeFile = "welcome.html";
   private static string _textWelcomeFile = "welcome.txt";
   private static string _couponsDataFile = "sample_coupons.json";

   // Relative location of the shared workflow resources folder.
   private static string _resourcesFilePathLocation = "../../../../../../../
workflows/sesv2_weekly_mailer/resources/";

   public static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonSimpleEmailServiceV2>()
                   .AddTransient<SESv2Wrapper>()
           )
           .Build();
```

```
        ServicesSetup(host);

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Amazon SES v2 Coupon Newsletter
Workflow.");
            Console.WriteLine("This workflow demonstrates how to use the Amazon
Simple Email Service (SES) v2 " +
                              "\r\nto send a coupon newsletter to a list of
subscribers.");

            // Prepare the application.
            var emailIdentity = await PrepareApplication();

            // Gather subscriber email addresses.
            await GatherSubscriberEmailAddresses(emailIdentity);

            // Send the coupon newsletter.
            await SendCouponNewsletter(emailIdentity);

            // Monitor and review.
            MonitorAndReview(true);

            // Clean up resources.
            await Cleanup(emailIdentity, true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon SES v2 Coupon Newsletter Workflow is
complete.");
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
```

```
    private static void ServicesSetup(IHost host)
    {
        _sesv2Wrapper = host.Services.GetRequiredService<SESv2Wrapper>();
    }

    /// <summary>
    /// Set up the resources for the workflow.
    /// </summary>
    /// <returns>The email address of the verified identity.</returns>
    public static async Task<string?> PrepareApplication()
    {
        var htmlContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_htmlContentFile);
        var textContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_textContentFile);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("1. In this step, we will prepare the application:" +
                          "\r\n  - Create a verified email identity for sending and
replying to emails." +
                          "\r\n  - Create a contact list to store the subscribers'
email addresses." +
                          "\r\n  - Create an email template for the coupon
newsletter.\r\n");

        // Prompt the user for a verified email address.
        while (!IsEmail(_verifiedEmail))
        {
            Console.Write("Enter a verified email address or an email to verify: ");
            _verifiedEmail = Console.ReadLine();
        }

        try
        {
            // Create an email identity and start the verification process.
            await _sesv2Wrapper.CreateEmailIdentityAsync(_verifiedEmail);
            Console.WriteLine($"Identity {_verifiedEmail} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Identity {_verifiedEmail} already exists.");
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"Error creating email identity: {ex.Message}");
        }

        // Create a contact list.
        try
        {
            await _sesv2Wrapper.CreateContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Contact list {_contactListName} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating contact list: {ex.Message}");
        }

        // Create an email template.
        try
        {
            await _sesv2Wrapper.CreateEmailTemplateAsync(_templateName, _subject,
htmlContent, textContent);
            Console.WriteLine($"Email template {_templateName} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Email template {_templateName} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating email template: {ex.Message}");
        }

        return _verifiedEmail;
    }

    /// <summary>
    /// Generate subscriber addresses and send welcome emails.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
```

```
    public static async Task<bool> GatherSubscriberEmailAddresses(string
 fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("2. In Step 2, we will gather subscriber email addresses:"
 +
                          "\r\n  - Prompt the user for a base email address." +
                          "\r\n  - Create 3 variants of the email address using
 subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com)." +
                          "\r\n  - Add each variant as a contact to the contact
 list." +
                          "\r\n  - Send a welcome email to each new contact.\r\n");

        // Prompt the user for a base email address.
        while (!IsEmail(_baseEmailAddress))
        {
            Console.Write("Enter a base email address (e.g., user@example.com): ");
            _baseEmailAddress = Console.ReadLine();
        }

        // Create 3 variants of the email address using +ses-weekly-newsletter-1,
 +ses-weekly-newsletter-2, etc.
        var baseEmailAddressParts = _baseEmailAddress!.Split("@");
        for (int i = 1; i <= 3; i++)
        {
            string emailAddress = $"{baseEmailAddressParts[0]}+ses-weekly-
newsletter-{i}@{baseEmailAddressParts[1]}";

            try
            {
                // Create a contact with the email address in the contact list.
                await _sesv2Wrapper.CreateContactAsync(emailAddress,
 _contactListName);
                Console.WriteLine($"Contact {emailAddress} added to the
 {_contactListName} contact list.");
            }
            catch (AlreadyExistsException)
            {
                Console.WriteLine($"Contact {emailAddress} already exists in the
 {_contactListName} contact list.");
            }
            catch (Exception ex)
            {
```

```
                Console.WriteLine($"Error creating contact {emailAddress}:
{ex.Message}");
                return false;
            }

            // Send a welcome email to the new contact.
            try
            {
                string subject = "Welcome to the Weekly Coupons Newsletter";
                string htmlContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _htmlWelcomeFile);
                string textContent = await
File.ReadAllTextAsync(_resourcesFilePathLocation + _textWelcomeFile);

                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress, new
List<string> { emailAddress }, subject, htmlContent, textContent);
                Console.WriteLine($"Welcome email sent to {emailAddress}.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error sending welcome email to {emailAddress}:
{ex.Message}");
                return false;
            }

            // Wait 2 seconds before sending the next email (if the account is in
the SES Sandbox).
            await Task.Delay(2000);
        }

        return true;
    }

    /// <summary>
    ///  Send the coupon newsletter to the subscribers in the contact list.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> SendCouponNewsletter(string fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("3. In this step, we will send the coupon newsletter:" +
```

```
                        "\r\n  - Retrieve the list of contacts from the contact
list." +
                        "\r\n  - Send the coupon newsletter using the email
template to each contact.\r\n");


       // Retrieve the list of contacts from the contact list.
       var contacts = await _sesv2Wrapper.ListContactsAsync(_contactListName);
       if (!contacts.Any())
       {
           Console.WriteLine($"No contacts found in the {_contactListName} contact
list.");
           return false;
       }

       // Load the coupon data from the sample_coupons.json file.
       string couponsData = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _couponsDataFile);

       // Send the coupon newsletter to each contact using the email template.
       try
       {
           foreach (var contact in contacts)
           {
               // To use the Contact List for list management, send to only one
address at a time.
               await _sesv2Wrapper.SendEmailAsync(fromEmailAddress,
                   new List<string> { contact.EmailAddress },
                   null, null, null, _templateName, couponsData, _contactListName);
           }

           Console.WriteLine($"Coupon newsletter sent to contact list
{_contactListName}.");
       }
       catch (Exception ex)
       {
           Console.WriteLine($"Error sending coupon newsletter to contact list
{_contactListName}: {ex.Message}");
           return false;
       }

       return true;
   }
```

```
    /// <summary>
    /// Provide instructions for monitoring sending activity and metrics.
    /// </summary>
    /// <param name="interactive">True to run in interactive mode.</param>
    /// <returns>True if successful.</returns>
    public static bool MonitorAndReview(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. In step 4, we will monitor and review:" +
                          "\r\n  - Provide instructions for the user to review the
sending activity and metrics in the AWS console.\r\n");

       Console.WriteLine("Review your sending activity using the SES Homepage in
the AWS console.");
       Console.WriteLine("Press Enter to open the SES Homepage in your default
browser...");
       if (interactive)
       {
           Console.ReadLine();
           try
           {
               // Open the SES Homepage in the default browser.
               Process.Start(new ProcessStartInfo
               {
                   FileName = "https://console.aws.amazon.com/ses/home",
                   UseShellExecute = true
               });
           }
           catch (Exception ex)
           {
               Console.WriteLine($"Error opening the SES Homepage: {ex.Message}");
               return false;
           }
       }

       Console.WriteLine("Review the sending activity and email metrics, then press
Enter to continue...");
       if (interactive)
           Console.ReadLine();
       return true;
    }

    /// <summary>
    /// Clean up the resources used in the workflow.
```

```
    /// </summary>
    /// <param name="verifiedEmailAddress">The verified email address from
 PrepareApplication.</param>
    /// <param name="interactive">True if interactive.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Cleanup(string verifiedEmailAddress, bool
 interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Finally, we clean up resources:" +
                          "\r\n  - Delete the contact list (which also deletes all
 contacts within it)." +
                          "\r\n  - Delete the email template." +
                          "\r\n  - Optionally delete the verified email identity.\r
\n");

        Console.WriteLine("Cleaning up resources...");

        // Delete the contact list (this also deletes all contacts in the list).
        try
        {
            await _sesv2Wrapper.DeleteContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Contact list {_contactListName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting contact list {_contactListName}:
 {ex.Message}");
            return false;
        }

        // Delete the email template.
        try
        {
            await _sesv2Wrapper.DeleteEmailTemplateAsync(_templateName);
            Console.WriteLine($"Email template {_templateName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Email template {_templateName} not found.");
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting email template {_templateName}:
{ex.Message}");
            return false;
        }

        // Ask the user if they want to delete the email identity.
        var deleteIdentity = !interactive ||
            GetYesNoResponse(
                $"Do you want to delete the email identity {verifiedEmailAddress}?
(y/n) ");
        if (deleteIdentity)
        {
            try
            {
                await _sesv2Wrapper.DeleteEmailIdentityAsync(verifiedEmailAddress);
                Console.WriteLine($"Email identity {verifiedEmailAddress}
deleted.");
            }
            catch (NotFoundException)
            {
                Console.WriteLine(
                    $"Email identity {verifiedEmailAddress} not found.");
            }
            catch (Exception ex)
            {
                Console.WriteLine(
                    $"Error deleting email identity {verifiedEmailAddress}:
{ex.Message}");
                return false;
            }
        }
        else
        {
            Console.WriteLine(
                $"Skipping deletion of email identity {verifiedEmailAddress}.");
        }

        return true;
    }

    /// <summary>
```

```
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Simple check to verify a string is an email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email.</returns>
    private static bool IsEmail(string? email)
    {
        if (string.IsNullOrEmpty(email))
            return false;
        return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$",
 RegexOptions.IgnoreCase);
    }
}
```

Envoltorio para operaciones de servicio.

```
using System.Net;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;

namespace Sesv2Scenario;

/// <summary>
/// Wrapper class for Amazon Simple Email Service (SES) v2 operations.
/// </summary>
public class SESv2Wrapper
{

    private readonly IAmazonSimpleEmailServiceV2 _sesClient;
```

```
    /// <summary>
    /// Constructor for the SESv2Wrapper.
    /// </summary>
    /// <param name="sesClient">The injected SES v2 client.</param>
    public SESv2Wrapper(IAmazonSimpleEmailServiceV2 sesClient)
    {
        _sesClient = sesClient;
    }

    /// <summary>
    /// Creates a contact and adds it to the specified contact list.
    /// </summary>
    /// <param name="emailAddress">The email address of the contact.</param>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The response from the CreateContact operation.</returns>
    public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
    {
        var request = new CreateContactRequest
        {
            EmailAddress = emailAddress,
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
```

```
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
```

```
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates an email identity (email address or domain) and starts the
verification process.
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
    /// <returns>The response from the CreateEmailIdentity operation.</returns>
    public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
    {
        var request = new CreateEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.CreateEmailIdentityAsync(request);
            return response;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email identity {emailIdentity} already exists.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
            throw;
```

```
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email identities has been exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
```

```
            {
                Subject = subject,
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
 exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email template:
 {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteContactListAsync(string contactListName)
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email template:
 {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };
```

```
        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
 {ex.Message}");
        }

        return new List<Contact>();
    }

    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email from.</
param>
    /// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
    /// <param name="subject">The subject of the email.</param>
    /// <param name="htmlContent">The HTML content of the email.</param>
    /// <param name="textContent">The text content of the email.</param>
    /// <param name="templateName">The name of the email template to use
 (optional).</param>
    /// <param name="templateData">The data to replace placeholders in the email
 template (optional).</param>
    /// <param name="contactListName">The name of the contact list for unsubscribe
 functionality (optional).</param>
    /// <returns>The MessageId response from the SendEmail operation.</returns>
```

```
   public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,
       string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
   {
       var request = new SendEmailRequest
       {
           FromEmailAddress = fromEmailAddress
       };

       if (toEmailAddresses.Any())
       {
           request.Destination = new Destination { ToAddresses =
toEmailAddresses };
       }

       if (!string.IsNullOrEmpty(templateName))
       {
           request.Content = new EmailContent()
           {
               Template = new Template
               {
                   TemplateName = templateName,
                   TemplateData = templateData
               }
           };
       }
       else
       {
           request.Content = new EmailContent
           {
               Simple = new Message
               {
                   Subject = new Content { Data = subject },
                   Body = new Body
                   {
                       Html = new Content { Data = htmlContent },
                       Text = new Content { Data = textContent }
                   }
               }
           };
       }

       if (!string.IsNullOrEmpty(contactListName))
```

```
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
        }
        catch (MailFromDomainNotVerifiedException ex)
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
```

```
        }

        return string.Empty;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList
  - DeleteEmailIdentity
  - DeleteEmailTemplate
  - ListContacts
  - SendEmail.simple
  - SendEmail.plantilla

SNSEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonSNS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola Amazon SNS

Los siguientes ejemplos de código muestran cómo empezar a utilizar AmazonSNS.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"\tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListTopicsla AWS SDK for .NET APIReferencia.

Temas

Acciones

`CheckIfPhoneNumberIsOptedOut`

En el siguiente ejemplo de código, se muestra cómo usar `CheckIfPhoneNumberIsOptedOut`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// This example shows how to use the Amazon Simple Notification Service
    /// (Amazon SNS) to check whether a phone number has been opted out.
    /// </summary>
    public class IsPhoneNumOptedOut
    {
        public static async Task Main()
        {
            string phoneNumber = "+15551112222";

            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await CheckIfOptedOutAsync(client, phoneNumber);
        }
```

```
        /// <summary>
        /// Checks to see if the supplied phone number has been opted out.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS Client object used
        /// to check if the phone number has been opted out.</param>
        /// <param name="phoneNumber">A string representing the phone number
        /// to check.</param>
        public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
        {
            var request = new CheckIfPhoneNumberIsOptedOutRequest
            {
                PhoneNumber = phoneNumber,
            };

            try
            {
                var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    string optOutStatus = response.IsOptedOut ? "opted out" : "not
opted out.";
                    Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
                }
            }
            catch (AuthorizationErrorException ex)
            {
                Console.WriteLine($"{ex.Message}");
            }
        }
    }
```

- Para API obtener más información, consulte CheckIfPhoneNumberIsOptedOutla AWS SDK for .NET APIReferencia.

`CreateTopic`

En el siguiente ejemplo de código, se muestra cómo usar `CreateTopic`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Cree un tema con un nombre específico.

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

    /// <summary>
    /// This example shows how to use Amazon Simple Notification Service
    /// (Amazon SNS) to add a new Amazon SNS topic.
    /// </summary>
    public class CreateSNSTopic
    {
        public static async Task Main()
        {
            string topicName = "ExampleSNSTopic";

            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            var topicArn = await CreateSNSTopicAsync(client, topicName);
            Console.WriteLine($"New topic ARN: {topicArn}");
        }

        /// <summary>
        /// Creates a new SNS topic using the supplied topic name.
        /// </summary>
        /// <param name="client">The initialized SNS client object used to
        /// create the new topic.</param>
        /// <param name="topicName">A string representing the topic name.</param>
        /// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>
        public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
        {
            var request = new CreateTopicRequest
```

```
            {
                Name = topicName,
            };

            var response = await client.CreateTopicAsync(request);

            return response.TopicArn;
        }
    }
```

Cree un tema nuevo con un nombre y atributos específicos FIFO y de deduplicación.

```
    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
```

```
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }
```

- Para API obtener más información, consulte CreateTopicla AWS SDK for .NET APIReferencia.

`DeleteTopic`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTopic`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar un tema por su temaARN.

```
    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
```

```
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteTopicla AWS SDK for .NET APIReferencia.

`GetTopicAttributes`

En el siguiente ejemplo de código, se muestra cómo usar `GetTopicAttributes`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;

    /// <summary>
    /// This example shows how to retrieve the attributes of an Amazon Simple
    /// Notification Service (Amazon SNS) topic.
    /// </summary>
    public class GetTopicAttributes
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-west-2:000000000000:ExampleSNSTopic";
            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            var attributes = await GetTopicAttributesAsync(client, topicArn);
            DisplayTopicAttributes(attributes);
        }

        /// <summary>
```

```
        /// Given the ARN of the Amazon SNS topic, this method retrieves the topic
        /// attributes.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the attributes for the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic for which to retrieve
        /// the attributes.</param>
        /// <returns>A Dictionary of topic attributes.</returns>
        public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            var response = await client.GetTopicAttributesAsync(topicArn);

            return response.Attributes;
        }

        /// <summary>
        /// This method displays the attributes for an Amazon SNS topic.
        /// </summary>
        /// <param name="topicAttributes">A Dictionary containing the
        /// attributes for an Amazon SNS topic.</param>
        public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
        {
            foreach (KeyValuePair<string, string> entry in topicAttributes)
            {
                Console.WriteLine($"{entry.Key}: {entry.Value}\n");
            }
        }
    }
```

- Para API obtener más información, consulte GetTopicAttributesla AWS SDK for .NET APIReferencia.

`ListSubscriptions`

En el siguiente ejemplo de código, se muestra cómo usar `ListSubscriptions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   using Amazon.SimpleNotificationService;
   using Amazon.SimpleNotificationService.Model;

   /// <summary>
   /// This example will retrieve a list of the existing Amazon Simple
   /// Notification Service (Amazon SNS) subscriptions.
   /// </summary>
   public class ListSubscriptions
   {
       public static async Task Main()
       {
           IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

           Console.WriteLine("Enter a topic ARN to list subscriptions for a
specific topic, " +
                             "or press Enter to list subscriptions for all
topics.");
           var topicArn = Console.ReadLine();
           Console.WriteLine();

           var subscriptions = await GetSubscriptionsListAsync(client, topicArn);

           DisplaySubscriptionList(subscriptions);
       }

       /// <summary>
       /// Gets a list of the existing Amazon SNS subscriptions, optionally by
specifying a topic ARN.
       /// </summary>
       /// <param name="client">The initialized Amazon SNS client object used
```

```
        /// to obtain the list of subscriptions.</param>
        /// <param name="topicArn">The optional ARN of a specific topic. Defaults to
 null.</param>
        /// <returns>A list containing information about each subscription.</
returns>
        public static async Task<List<Subscription>>
 GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string topicArn
 = null)
        {
            var results = new List<Subscription>();

            if (!string.IsNullOrEmpty(topicArn))
            {
                var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
                    new ListSubscriptionsByTopicRequest()
                    {
                        TopicArn = topicArn,
                    });

                // Get the entire list using the paginator.
                await foreach (var subscription in paginateByTopic.Subscriptions)
                {
                    results.Add(subscription);
                }
            }
            else
            {
                var paginateAllSubscriptions =
 client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());

                // Get the entire list using the paginator.
                await foreach (var subscription in
 paginateAllSubscriptions.Subscriptions)
                {
                    results.Add(subscription);
                }
            }

            return results;
        }

        /// <summary>
        /// Display a list of Amazon SNS subscription information.
        /// </summary>
```

```
        /// <param name="subscriptionList">A list containing details for existing
        /// Amazon SNS subscriptions.</param>
        public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
        {
            foreach (var subscription in subscriptionList)
            {
                Console.WriteLine($"Owner: {subscription.Owner}");
                Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
                Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
                Console.WriteLine($"Endpoint: {subscription.Endpoint}");
                Console.WriteLine($"Protocol: {subscription.Protocol}");
                Console.WriteLine();
            }
        }
    }
```

- Para API obtener más información, consulte ListSubscriptionsla AWS SDK for .NET APIReferencia.

`ListTopics`

En el siguiente ejemplo de código, se muestra cómo usar `ListTopics`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;
```

```
    /// <summary>
    /// Lists the Amazon Simple Notification Service (Amazon SNS)
    /// topics for the current account.
    /// </summary>
    public class ListSNSTopics
    {
        public static async Task Main()
        {
            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await GetTopicListAsync(client);
        }

        /// <summary>
        /// Retrieves the list of Amazon SNS topics in groups of up to 100
        /// topics.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the list of topics.</param>
        public static async Task GetTopicListAsync(IAmazonSimpleNotificationService
client)
        {
            // If there are more than 100 Amazon SNS topics, the call to
            // ListTopicsAsync will return a value to pass to the
            // method to retrieve the next 100 (or less) topics.
            string nextToken = string.Empty;

            do
            {
                var response = await client.ListTopicsAsync(nextToken);
                DisplayTopicsList(response.Topics);
                nextToken = response.NextToken;
            }
            while (!string.IsNullOrEmpty(nextToken));
        }

        /// <summary>
        /// Displays the list of Amazon SNS Topic ARNs.
        /// </summary>
        /// <param name="topicList">The list of Topic ARNs.</param>
        public static void DisplayTopicsList(List<Topic> topicList)
        {
            foreach (var topic in topicList)
```

```
            {
                Console.WriteLine($"{topic.TopicArn}");
            }
        }
    }
```

- Para API obtener más información, consulte ListTopicsla AWS SDK for .NET APIReferencia.

`Publish`

En el siguiente ejemplo de código, se muestra cómo usar `Publish`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Publique un mensaje en un tema.

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

    /// <summary>
    /// This example publishes a message to an Amazon Simple Notification
    /// Service (Amazon SNS) topic.
    /// </summary>
    public class PublishToSNSTopic
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-east-2:000000000000:ExampleSNSTopic";
            string messageText = "This is an example message to publish to the
ExampleSNSTopic.";
```

```
            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await PublishToTopicAsync(client, topicArn, messageText);
        }

        /// <summary>
        /// Publishes a message to an Amazon SNS topic.
        /// </summary>
        /// <param name="client">The initialized client object used to publish
        /// to the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic.</param>
        /// <param name="messageText">The text of the message.</param>
        public static async Task PublishToTopicAsync(
            IAmazonSimpleNotificationService client,
            string topicArn,
            string messageText)
        {
            var request = new PublishRequest
            {
                TopicArn = topicArn,
                Message = messageText,
            };

            var response = await client.PublishAsync(request);

            Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
        }
    }
```

Publique un mensaje en un tema con opciones de grupo, duplicación y atributo.

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");
```

```
        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
```

```
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
 messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?", false);
        }
    }
```

Aplica las selecciones del usuario a la acción de publicación.

```
    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
```

```
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
```

- Para API obtener más información, consulte Publicar en AWS SDK for .NET APIreferencia.

`Subscribe`

En el siguiente ejemplo de código, se muestra cómo usar `Subscribe`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Suscribe una dirección de correo electrónico a un tema.

```
        /// <summary>
        /// Creates a new subscription to a topic.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object, used
```

```
        /// to create an Amazon SNS subscription.</param>
        /// <param name="topicArn">The ARN of the topic to subscribe to.</param>
        /// <returns>A SubscribeResponse object which includes the subscription
        /// ARN for the new subscription.</returns>
        public static async Task<SubscribeResponse> TopicSubscribeAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            SubscribeRequest request = new SubscribeRequest()
            {
                TopicArn = topicArn,
                ReturnSubscriptionArn = true,
                Protocol = "email",
                Endpoint = "recipient@example.com",
            };

            var response = await client.SubscribeAsync(request);

            return response;
        }
```

Suscriba una cola a un tema con filtros opcionales.

```
    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };
```

```
        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }
```

- Para API obtener más información, consulte Suscríbete AWS SDK for .NET APIcomo referencia.

`Unsubscribe`

En el siguiente ejemplo de código, se muestra cómo usar `Unsubscribe`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Darse de baja de un tema mediante una suscripciónARN.

```
    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
```

```
    }
```

- Para API obtener más información, consulte Cancelar suscripción en AWS SDK for .NET APIReference.

Escenarios

Creación de una SNS aplicación de Amazon

El siguiente ejemplo de código muestra cómo crear una aplicación que tenga funciones de suscripción y publicación y que traduzca los mensajes.

AWS SDK for .NET

Muestra cómo utilizar el Amazon Simple Notification Service. NETAPIpara crear una aplicación web con funciones de suscripción y publicación. Además, esta aplicación de ejemplo también traduce los mensajes.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarla y ejecutarla, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon SNS
- Amazon Translate

Creación de una aplicación sin servidor para administrar fotos

En el siguiente ejemplo de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- APIPuerta de enlace
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Publica un mensaje SMS de texto

El siguiente ejemplo de código muestra cómo publicar SMS mensajes con AmazonSNS.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
        /// Initializes a new instance of the <see cref="SNSMessage"/> class.
        /// Constructs a new SNSMessage object initializing the Amazon Simple
        /// Notification Service (Amazon SNS) client using the supplied
        /// Region endpoint.
```

```
        /// </summary>
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
        /// sending test messages with this object.</param>
        public SNSMessage(RegionEndpoint regionEndpoint)
        {
            snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);
        }

        /// <summary>
        /// Sends the SMS message passed in the text parameter to the phone number
        /// in phoneNum.
        /// </summary>
        /// <param name="phoneNum">The ten-digit phone number to which the text
        /// message will be sent.</param>
        /// <param name="text">The text of the message to send.</param>
        /// <returns>Async task.</returns>
        public async Task SendTextMessageAsync(string phoneNum, string text)
        {
            if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
            {
                return;
            }

            // Now actually send the message.
            var request = new PublishRequest
            {
                Message = text,
                PhoneNumber = phoneNum,
            };

            try
            {
                var response = await snsClient.PublishAsync(request);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error sending message: {ex}");
            }
        }
    }
}
```

- Para API obtener más información, consulte Publicar en AWS SDK for .NET APIreferencia.

Publicación de mensajes en colas

En el siguiente ejemplo de código, se muestra cómo:

- Cree un tema (FIFOo noFIFO).
- Suscribir varias colas al tema con la opción de aplicar un filtro.
- Publicar mensajes en el tema.
- Sondear las colas en busca de los mensajes recibidos.

AWS SDK for .NET

> **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.

```
/// <summary>
/// Console application to run a workflow scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
```

```
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();
```

```
            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues workflow is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the workflow.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this workflow, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                          $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                          $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
 the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
 is supported." +
                              $"\r\nDeduplication IDs are either set in the message
 or automatically generated " +
                              $"\r\nfrom content using a hash function.\r\n" +
                              $"\r\nIf a message is successfully published to an SNS
 FIFO topic, any message " +
                              $"\r\npublished and determined to have the same
 deduplication ID, " +
                              $"\r\nwithin the five-minute deduplication interval,
 is accepted but not delivered.\r\n" +
                              $"\r\nFor more information about deduplication, " +
```

```
                                $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
 _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
 ", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
 appended to the queue name.");
                }
```

```
                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
    /// <returns>Async Task.</returns>
```

```
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
        }
```

```
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
```

```
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
```

```
                    Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                }

                var selection = GetUserResponse("", "1");
                int.TryParse(selection, out var selectionNumber);

                if (selectionNumber > 0 && selectionNumber < _tones.Length)
                {
                    toneAttribute = _tones[selectionNumber - 1];
                }
            }

            var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

            Console.WriteLine($"Message published with id {messageID}.");
        }

        keepSendingMessages = GetYesNoResponse("Send another message?", false);
    }
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                      "\r\nPress any key to continue.");
    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();
    while (moreMessages)
    {
        var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);
```

```
            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
```

```
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
    {
```

```
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Crea una clase que abarque las SQS operaciones de Amazon.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
```

```
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
 useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }
```

```
            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
```

```
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\": \"{topicArn}\""
+
                                            "}" +
                                       "}" +
                                "}]" +
                              "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
 queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
```

```
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
```

```
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Crea una clase que abarque las SNS operaciones de Amazon.

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
```

```
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
```

```
            subscribeRequest.Attributes = new Dictionary<string, string>
 { { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
 _amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
```

```
                    { attributeName!, new MessageAttributeValue() { StringValue =
 attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publicación
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Ejemplos sin servidor

Invocar una función Lambda desde un disparador de Amazon SNS

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de mensajes de un SNS tema. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un SNS evento con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
 context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

SQSEjemplos de Amazon que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante AmazonSQS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Hola Amazon SQS

Los siguientes ejemplos de código muestran cómo empezar a utilizar AmazonSQS.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();
```

```
        Console.WriteLine($"Hello Amazon SQS! Following are some of your queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- Para API obtener más información, consulte ListQueuesla AWS SDK for .NET APIReferencia.

Temas

- Acciones
- Escenarios
- Ejemplos sin servidor

Acciones

`CreateQueue`

En el siguiente ejemplo de código, se muestra cómo usar `CreateQueue`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola con un nombre específico.

```
    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
```

```
        return createResponse.QueueUrl;
    }
```

Crea una SQS cola de Amazon y envíale un mensaje.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
bestseller for week of 12/11/2016.";
```

```
            var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
 messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
 string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
```

```
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- Para API obtener más información, consulta CreateQueuela AWS SDK for .NET APIReferencia.

`DeleteMessage`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteMessage`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibe un mensaje de una SQS cola de Amazon y, a continuación, borra el mensaje.

```
        public static async Task Main()
```

```
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
```

```
        public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- Para API obtener más información, consulta la DeleteMessagesección AWS SDK for .NET APIReferencia.

`DeleteMessageBatch`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteMessageBatch`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }
```

- Para API obtener más información, consulte DeleteMessageBatchla AWS SDK for .NET APIReferencia.

`DeleteQueue`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteQueue`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Elimine una cola utilizando suURL.

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte DeleteQueuela AWS SDK for .NET APIReferencia.

`GetQueueAttributes`

En el siguiente ejemplo de código, se muestra cómo usar `GetQueueAttributes`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
        getAttributesRequest);

    return getAttributesResponse.QueueARN;
}
```

- Para API obtener más información, consulte GetQueueAttributesla AWS SDK for .NET APIReferencia.

`GetQueueUrl`

En el siguiente ejemplo de código, se muestra cómo usar `GetQueueUrl`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

   public class GetQueueUrl
   {
       /// <summary>
       /// Initializes the Amazon SQS client object and then calls the
       /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
       /// queue.
       /// </summary>
       public static async Task Main()
       {
           // If the Amazon SQS message queue is not in the same AWS Region as your
           // default user, you need to provide the AWS Region as a parameter to
the
           // client constructor.
           var client = new AmazonSQSClient();

           string queueName = "New-Example-Queue";

           try
           {
               var response = await client.GetQueueUrlAsync(queueName);

               if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
               {
                   Console.WriteLine($"The URL for {queueName} is:
{response.QueueUrl}");
               }
           }
           catch (QueueDoesNotExistException ex)
           {
```

```
                Console.WriteLine(ex.Message);
                Console.WriteLine($"The queue {queueName} was not found.");
            }
        }
    }
```

- Para API obtener más información, consulte GetQueueUrlla AWS SDK for .NET APIReferencia.

`ReceiveMessage`

En el siguiente ejemplo de código, se muestra cómo usar `ReceiveMessage`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Reciba mensajes de una cola utilizando suURL.

```
    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
```

```
            });
        return messageResponse.Messages;
    }
```

Recibe un mensaje de una SQS cola de Amazon y, a continuación, borra el mensaje.

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }
```

```
        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- Para API obtener más información, consulta la ReceiveMessagesección AWS SDK for .NET APIReferencia.

`SendMessage`

En el siguiente ejemplo de código, se muestra cómo usar `SendMessage`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crea una SQS cola de Amazon y envíale un mensaje.

```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   using Amazon;
   using Amazon.SQS;
   using Amazon.SQS.Model;

   public class CreateSendExample
   {
       // Specify your AWS Region (an example Region is shown).
       private static readonly string QueueName = "Example_Queue";
       private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
       private static IAmazonSQS client;

       public static async Task Main()
       {
           client = new AmazonSQSClient(ServiceRegion);
           var createQueueResponse = await CreateQueue(client, QueueName);

           string queueUrl = createQueueResponse.QueueUrl;

           Dictionary<string, MessageAttributeValue> messageAttributes = new
Dictionary<string, MessageAttributeValue>
           {
               { "Title",   new MessageAttributeValue { DataType = "String",
StringValue = "The Whistler" } },
               { "Author",  new MessageAttributeValue { DataType = "String",
StringValue = "John Grisham" } },
```

```
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
 messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
 string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
```

```
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- Para API obtener más información, consulta SendMessagela AWS SDK for .NET APIReferencia.

`SetQueueAttributes`

En el siguiente ejemplo de código, se muestra cómo usar `SetQueueAttributes`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Establecer el atributo de política de una cola para un tema.

```
    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\": \"{topicArn}\""
+
                                            "}" +
                                       "}" +
                                "}]" +
                            "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
```

```
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte SetQueueAttributesla AWS SDK for .NET APIReferencia.

Escenarios

Publicación de mensajes en colas

En el siguiente ejemplo de código, se muestra cómo:

- Cree un tema (FIFOo noFIFO).
- Suscribir varias colas al tema con la opción de aplicar un filtro.
- Publicar mensajes en el tema.
- Sondear las colas en busca de los mensajes recibidos.

AWS SDK for .NET

> **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.

```
/// <summary>
/// Console application to run a workflow scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
```

```
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }
```

```
    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues workflow is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the workflow.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this workflow, you will create an SNS topic and
 subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for configuring
 the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                          $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                          $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
 the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
 is supported." +
```

```
                                $"\r\nDeduplication IDs are either set in the message
 or automatically generated " +
                                $"\r\nfrom content using a hash function.\r\n" +
                                $"\r\nIf a message is successfully published to an SNS
 FIFO topic, any message " +
                                $"\r\npublished and determined to have the same
 deduplication ID, " +
                                $"\r\nwithin the five-minute deduplication interval,
 is accepted but not delivered.\r\n" +
                                $"\r\nFor more information about deduplication, " +
                                $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
 _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
 ", $"example-queue-{i}");
```

```
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
                }

                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
```

```
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
```

```
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");
```

```
                Console.WriteLine("Enter a deduplication ID for this message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
                Console.WriteLine("Enter a number for an attribute.");
                for (int i = 0; i < _tones.Length; i++)
                {
                    Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                }

                var selection = GetUserResponse("", "1");
                int.TryParse(selection, out var selectionNumber);

                if (selectionNumber > 0 && selectionNumber < _tones.Length)
                {
                    toneAttribute = _tones[selectionNumber - 1];
                }
            }

            var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

            Console.WriteLine($"Message published with id {messageID}.");
        }

        keepSendingMessages = GetYesNoResponse("Send another message?", false);
    }
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                      "\r\nPress any key to continue.");
```

```
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }
```

```
    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
```

```
    }
}
```

Crea una clase que abarque las SQS operaciones de Amazon.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
 useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
```

```
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }
```

```
   /// <summary>
   /// Set the policy attribute of a queue for a topic.
   /// </summary>
   /// <param name="queueArn">The ARN of the queue.</param>
   /// <param name="topicArn">The ARN of the topic.</param>
   /// <param name="queueUrl">The url for the queue.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
   {
       var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\": \"{topicArn}\""
+
                                                "}" +
                                          "}" +
                                "}]" +
                              "}";
       var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
           new SetQueueAttributesRequest()
           {
               QueueUrl = queueUrl,
               Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
           });
       return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
   }

   /// <summary>
   /// Receive messages from a queue by its URL.
   /// </summary>
   /// <param name="queueUrl">The url of the queue.</param>
   /// <returns>The list of messages.</returns>
```

```
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
 messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
 _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }
```

```
    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Crea una clase que abarque las SNS operaciones de Amazon.

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
```

```
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
 "true");
            }
        }

        var createResponse = await
 _amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
 subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
```

```
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
 filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
 { { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
 _amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
```

```
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
```

```
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- Para API obtener más información, consulte los siguientes temas en la sección de AWS SDK for .NET APIreferencia.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publicación
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Utilice el marco de procesamiento de AWS mensajes para. NETcon Amazon SQS

El siguiente ejemplo de código muestra cómo crear aplicaciones que publiquen y reciban SQS mensajes de Amazon mediante el marco de procesamiento de AWS mensajes para. NET.

AWS SDK for .NET

Proporciona un tutorial sobre el marco de procesamiento de AWS mensajes para. NET. El tutorial crea una aplicación web que permite al usuario publicar un SQS mensaje de Amazon y una aplicación de línea de comandos que recibe el mensaje.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el tutorial completo en la Guía para AWS SDK for .NET desarrolladores y el ejemplo correspondiente. GitHub

Servicios utilizados en este ejemplo

- Amazon SQS

Ejemplos sin servidor

Invocar una función Lambda desde un disparador de Amazon SQS

El siguiente ejemplo de código muestra cómo implementar una función Lambda que recibe un evento desencadenado por la recepción de mensajes de una SQS cola. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Consumir un SQS evento con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
```

```
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Notificación de errores de artículos de lotes para funciones de Lambda con un activador de Amazon SQS

El siguiente ejemplo de código muestra cómo implementar una respuesta por lotes parcial para las funciones de Lambda que reciben eventos de una SQS cola. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos sin servidor.

Cómo informar de los errores de los elementos del SQS lote con Lambda mediante. NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
```

```
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Ejemplos de Step Functions usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso de AWS SDK for .NET with Step Functions.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

Introducción a Step Functions

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Step Functions.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

public class HelloStepFunctions
{
    static async Task Main()
    {
        var stepFunctionsClient = new AmazonStepFunctionsClient();

        Console.Clear();
        Console.WriteLine("Welcome to AWS Step Functions");
        Console.WriteLine("Let's list up to 10 of your state machines:");
        var stateMachineListRequest = new ListStateMachinesRequest { MaxResults =
 10 };

        // Get information for up to 10 Step Functions state machines.
        var response = await
 stepFunctionsClient.ListStateMachinesAsync(stateMachineListRequest);

        if (response.StateMachines.Count > 0)
        {
            response.StateMachines.ForEach(stateMachine =>
            {
                Console.WriteLine($"State Machine Name: {stateMachine.Name}\tAmazon
 Resource Name (ARN): {stateMachine.StateMachineArn}");
            });
        }
        else
        {
            Console.WriteLine("\tNo state machines were found.");
        }
```

```
    }
}
```

- Para API obtener más información, consulte ListStateMachinesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Crear una actividad
- Crear una máquina de estado a partir de una definición de Amazon States Language que contenga la actividad creada anteriormente como un paso
- Ejecutar la máquina de estados y responder a la actividad con entradas de usuario
- Obtener el resultado y el estado final una vez completada la ejecución y, luego, limpiar los recursos.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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.

```
global using System.Text.Json;
global using Amazon.StepFunctions;
```

```
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using StepFunctionsActions;
global using LogLevel = Microsoft.Extensions.Logging.LogLevel;



using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.StepFunctions.Model;

namespace StepFunctionsBasics;

public class StepFunctionsBasics
{
    private static ILogger _logger = null!;
    private static IConfigurationRoot _configuration = null!;
    private static IAmazonIdentityManagementService _iamService = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Step Functions.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonStepFunctions>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<StepFunctionsWrapper>()
            )
            .Build();

        _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<StepFunctionsBasics>();

        // Load configuration settings.
        _configuration = new ConfigurationBuilder()
```

```
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var activityName = _configuration["ActivityName"];
        var stateMachineName = _configuration["StateMachineName"];

        var roleName = _configuration["RoleName"];
        var repoBaseDir = _configuration["RepoBaseDir"];
        var jsonFilePath = _configuration["JsonFilePath"];
        var jsonFileName = _configuration["JsonFileName"];

        var uiMethods = new UiMethods();
        var stepFunctionsWrapper =
 host.Services.GetRequiredService<StepFunctionsWrapper>();

        _iamService =
 host.Services.GetRequiredService<IAmazonIdentityManagementService>();

        // Load definition for the state machine from a JSON file.
        var stateDefinitionJson = File.ReadAllText($"{repoBaseDir}{jsonFilePath}
{jsonFileName}");

        Console.Clear();
        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Create activity");
        Console.WriteLine("Let's start by creating an activity.");
        string activityArn;
        string stateMachineArn;

        // Check to see if the activity already exists.
        var activityList = await stepFunctionsWrapper.ListActivitiesAsync();
        var existingActivity = activityList.FirstOrDefault(activity => activity.Name
 == activityName);
        if (existingActivity is not null)
        {
            activityArn = existingActivity.ActivityArn;
            Console.WriteLine($"Activity, {activityName}, already exists.");
        }
        else
```

```
        {
            activityArn = await stepFunctionsWrapper.CreateActivity(activityName);
        }

        // Swap the placeholder in the JSON file with the Amazon Resource Name (ARN)
        // of the recently created activity.
        var stateDefinition =
stateDefinitionJson.Replace("{{DOC_EXAMPLE_ACTIVITY_ARN}}", activityArn);

        uiMethods.DisplayTitle("Create state machine");
        Console.WriteLine("Now we'll create a state machine.");

        // Find or create an IAM role that can be assumed by Step Functions.
        var role = await GetOrCreateStateMachineRole(roleName);

        // See if the state machine already exists.
        var stateMachineList = await stepFunctionsWrapper.ListStateMachinesAsync();
        var existingStateMachine =
            stateMachineList.FirstOrDefault(stateMachine => stateMachine.Name ==
stateMachineName);
        if (existingStateMachine is not null)
        {
            Console.WriteLine($"State machine, {stateMachineName}, already
exists.");
            stateMachineArn = existingStateMachine.StateMachineArn;
        }
        else
        {
            // Create the state machine.
            stateMachineArn =
                await stepFunctionsWrapper.CreateStateMachine(stateMachineName,
stateDefinition, role.Arn);
            uiMethods.PressEnter();
        }

        Console.WriteLine("The state machine has been created.");
        var describeStateMachineResponse = await
stepFunctionsWrapper.DescribeStateMachineAsync(stateMachineArn);


Console.WriteLine($"{describeStateMachineResponse.Name}\t{describeStateMachineResponse.Stat
        Console.WriteLine($"Current status: {describeStateMachineResponse.Status}");
        Console.WriteLine($"Amazon Resource Name (ARN) of the role assumed by the
state machine: {describeStateMachineResponse.RoleArn}");
```

```
        var userName = string.Empty;
        Console.Write("Before we start the state machine, tell me what should
ChatSFN call you? ");
        userName = Console.ReadLine();

        // Keep asking until the user enters a string value.
        while (string.IsNullOrEmpty(userName))
        {
            Console.Write("Enter your name: ");
            userName = Console.ReadLine();
        }

        var executionJson = @"{""name"": """ + userName + @"""}";

        // Start the state machine execution.
        Console.WriteLine("Now we'll start execution of the state machine.");
        var executionArn = await
stepFunctionsWrapper.StartExecutionAsync(executionJson, stateMachineArn);
        Console.WriteLine("State machine started.");

        Console.WriteLine($"Thank you, {userName}. Now let's get started...");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("ChatSFN");

        var isDone = false;
        var response = new GetActivityTaskResponse();
        var taskToken = string.Empty;
        var userChoice = string.Empty;

        while (!isDone)
        {
            response = await stepFunctionsWrapper.GetActivityTaskAsync(activityArn,
"MvpWorker");
            taskToken = response.TaskToken;

            // Parse the returned JSON string.
            var taskJsonResponse = JsonDocument.Parse(response.Input);
            var taskJsonObject = taskJsonResponse.RootElement;
            var message = taskJsonObject.GetProperty("message").GetString();
            var actions =
taskJsonObject.GetProperty("actions").EnumerateArray().Select(x =>
x.ToString()).ToList();
```

```
            Console.WriteLine($"\n{message}\n");

            // Prompt the user for another choice.
            Console.WriteLine("ChatSFN: What would you like me to do?");
            actions.ForEach(action => Console.WriteLine($"\t{action}"));
            Console.Write($"\n{userName}, tell me your choice: ");
            userChoice = Console.ReadLine();
            if (userChoice?.ToLower() == "done")
            {
                isDone = true;
            }

            Console.WriteLine($"You have selected: {userChoice}");
            var jsonResponse = @"{""action"": """ + userChoice + @"""}";

            await stepFunctionsWrapper.SendTaskSuccessAsync(taskToken,
jsonResponse);
        }

        await stepFunctionsWrapper.StopExecution(executionArn);
        Console.WriteLine("Now we will wait for the execution to stop.");
        DescribeExecutionResponse executionResponse;
        do
        {
            executionResponse = await
stepFunctionsWrapper.DescribeExecutionAsync(executionArn);
        } while (executionResponse.Status == ExecutionStatus.RUNNING);

        Console.WriteLine("State machine stopped.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("State machine executions");
        Console.WriteLine("Now let's take a look at the execution values for the
state machine.");

        // List the executions.
        var executions = await
stepFunctionsWrapper.ListExecutionsAsync(stateMachineArn);

        uiMethods.DisplayTitle("Step function execution values");
        executions.ForEach(execution =>
        {
            Console.WriteLine($"{execution.Name}\t{execution.StartDate} to
{execution.StopDate}");
```

```
        });

        uiMethods.PressEnter();

        // Now delete the state machine and the activity.
        uiMethods.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the state machine...");

        await stepFunctionsWrapper.DeleteStateMachine(stateMachineArn);
        Console.WriteLine("State machine deleted.");

        Console.WriteLine("Deleting the activity...");
        await stepFunctionsWrapper.DeleteActivity(activityArn);
        Console.WriteLine("Activity deleted.");

        Console.WriteLine("The Amazon Step Functions scenario is now complete.");
    }

    static async Task<Role> GetOrCreateStateMachineRole(string roleName)
    {
        // Define the policy document for the role.
        var stateMachineRolePolicy = @"{
         ""Version"": ""2012-10-17"",
        ""Statement"": [{
            ""Sid"": """",
            ""Effect"": ""Allow"",
            ""Principal"": {
                ""Service"": ""states.amazonaws.com""},
            ""Action"": ""sts:AssumeRole""}]}";

        var role = new Role();
        var roleExists = false;

        try
        {
            var getRoleResponse = await _iamService.GetRoleAsync(new GetRoleRequest
{ RoleName = roleName });
            roleExists = true;
            role = getRoleResponse.Role;
        }
        catch (NoSuchEntityException)
        {
            // The role doesn't exist. Create it.
            Console.WriteLine($"Role, {roleName} doesn't exist. Creating it...");
```

```
        }

        if (!roleExists)
        {
            var request = new CreateRoleRequest
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = stateMachineRolePolicy,
            };

            var createRoleResponse = await _iamService.CreateRoleAsync(request);
            role = createRoleResponse.Role;
        }

        return role;
    }
}


namespace StepFunctionsBasics;

/// <summary>
/// Some useful methods to make screen display easier.
/// </summary>
public class UiMethods
{
    private readonly string _sepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Welcome to the AWS Step Functions Demo");

        Console.WriteLine("This example application will do the following:");
        Console.WriteLine("\t 1. Create an activity.");
        Console.WriteLine("\t 2. Create a state machine.");
        Console.WriteLine("\t 3. Start an execution.");
        Console.WriteLine("\t 4. Run the worker, then stop it.");
        Console.WriteLine("\t 5. List executions.");
        Console.WriteLine("\t 6. Clean up the resources created for the example.");
    }
```

```
    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue.");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter"></param>
    /// <returns></returns>
    private string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title, and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(_sepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(_sepBar);
    }
}
```

Definir una clase que contenga las acciones de máquina de estado y actividad.

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
```

```
using Amazon.StepFunctions.Model;

/// <summary>
/// Wrapper that performs AWS Step Functions actions.
/// </summary>
public class StepFunctionsWrapper
{
    private readonly IAmazonStepFunctions _amazonStepFunctions;

    /// <summary>
    /// The constructor for the StepFunctionsWrapper. Initializes the
    /// client object passed to it.
    /// </summary>
    /// <param name="amazonStepFunctions">An initialized Step Functions client
 object.</param>
    public StepFunctionsWrapper(IAmazonStepFunctions amazonStepFunctions)
    {
        _amazonStepFunctions = amazonStepFunctions;
    }

    /// <summary>
    /// Create a Step Functions activity using the supplied name.
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
 CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }


    /// <summary>
    /// Create a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    /// machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    /// state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
```

```
   public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
   {
       var request = new CreateStateMachineRequest
       {
           Name = stateMachineName,
           Definition = definition,
           RoleArn = roleArn
       };

       var response =
           await _amazonStepFunctions.CreateStateMachineAsync(request);
       return response.StateMachineArn;
   }


   /// <summary>
   /// Delete a Step Machine activity.
   /// </summary>
   /// <param name="activityArn">The Amazon Resource Name (ARN) of
   /// the activity.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteActivity(string activityArn)
   {
       var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }


   /// <summary>
   /// Delete a Step Functions state machine.
   /// </summary>
   /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
   /// state machine.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteStateMachine(string stateMachineArn)
   {
       var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
       { StateMachineArn = stateMachineArn });
       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }
```

```
    /// <summary>
    /// Retrieve information about the specified Step Functions execution.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of the
    /// Step Functions execution.</param>
    /// <returns>The API response returned by the API.</returns>
    public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
 executionArn)
    {
        var response = await _amazonStepFunctions.DescribeExecutionAsync(new
 DescribeExecutionRequest { ExecutionArn = executionArn });
        return response;
    }


    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
 StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
 DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }


    /// <summary>
    /// Retrieve a task with the specified Step Functions activity
    /// with the specified Amazon Resource Name (ARN).
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the Step Functions activity.</param>
    /// <param name="workerName">The name of the Step Functions worker.</param>
    /// <returns>The response from the Step Functions activity.</returns>
    public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
 activityArn, string workerName)
    {
```

```
        var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
        { ActivityArn = activityArn, WorkerName = workerName });
        return response;
    }


    /// <summary>
    /// List the Step Functions activities for the current account.
    /// </summary>
    /// <returns>A list of ActivityListItems.</returns>
    public async Task<List<ActivityListItem>> ListActivitiesAsync()
    {
        var request = new ListActivitiesRequest();
        var activities = new List<ActivityListItem>();

        do
        {
            var response = await _amazonStepFunctions.ListActivitiesAsync(request);

            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }

            activities.AddRange(response.Activities);
        }
        while (request.NextToken is not null);

        return activities;
    }


    /// <summary>
    /// Retrieve information about executions of a Step Functions
    /// state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>A list of ExecutionListItem objects.</returns>
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
```

```
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }


    /// <summary>
    /// Retrieve a list of Step Functions state machines.
    /// </summary>
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }


    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
```

```
    /// <param name="taskResponse">The response received from executing the task.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
 taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
 SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
 stateMachineArn)
    {
        var executionRequest = new StartExecutionRequest
        {
            Input = executionJson,
            StateMachineArn = stateMachineArn
        };

        var response = await
 _amazonStepFunctions.StartExecutionAsync(executionRequest);
        return response.ExecutionArn;
    }


    /// <summary>
    /// Stop execution of a Step Functions workflow.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of
    /// the Step Functions execution to stop.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> StopExecution(string executionArn)
    {
        var response =
            await _amazonStepFunctions.StopExecutionAsync(new StopExecutionRequest
{ ExecutionArn = executionArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - CreateActivity
  - CreateStateMachine
  - DeleteActivity
  - DeleteStateMachine
  - DescribeExecution
  - DescribeStateMachine
  - GetActivityTask
  - ListActivities
  - ListStateMachines
  - SendTaskSuccess
  - StartExecution
  - StopExecution

Acciones

`CreateActivity`

En el siguiente ejemplo de código, se muestra cómo usar `CreateActivity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a Step Functions activity using the supplied name.
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
 CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }
```

- Para API obtener más información, consulte CreateActivityla AWS SDK for .NET APIReferencia.

`CreateStateMachine`

En el siguiente ejemplo de código, se muestra cómo usar `CreateStateMachine`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a Step Functions state machine.
```

```
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    /// machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    /// state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
    public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
    {
        var request = new CreateStateMachineRequest
        {
            Name = stateMachineName,
            Definition = definition,
            RoleArn = roleArn
        };

        var response =
            await _amazonStepFunctions.CreateStateMachineAsync(request);
        return response.StateMachineArn;
    }
```

- Para API obtener más información, consulte CreateStateMachinela AWS SDK for .NET APIReferencia.

`DeleteActivity`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteActivity`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a Step Machine activity.
```

```
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the activity.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteActivity(string activityArn)
    {
        var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteActivityla AWS SDK for .NET APIReferencia.

`DeleteStateMachine`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteStateMachine`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// state machine.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteStateMachine(string stateMachineArn)
    {
        var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
        { StateMachineArn = stateMachineArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteStateMachinela AWS SDK for .NET APIReferencia.

`DescribeExecution`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeExecution`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Retrieve information about the specified Step Functions execution.
   /// </summary>
   /// <param name="executionArn">The Amazon Resource Name (ARN) of the
   /// Step Functions execution.</param>
   /// <returns>The API response returned by the API.</returns>
   public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
   {
       var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
       return response;
   }
```

- Para API obtener más información, consulte DescribeExecutionla AWS SDK for .NET APIReferencia.

`DescribeStateMachine`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeStateMachine`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
 StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
 DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }
```

- Para API obtener más información, consulte DescribeStateMachinela AWS SDK for .NET APIReferencia.

`GetActivityTask`

En el siguiente ejemplo de código, se muestra cómo usar `GetActivityTask`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Retrieve a task with the specified Step Functions activity
    /// with the specified Amazon Resource Name (ARN).
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the Step Functions activity.</param>
    /// <param name="workerName">The name of the Step Functions worker.</param>
    /// <returns>The response from the Step Functions activity.</returns>
    public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
    {
        var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
        { ActivityArn = activityArn, WorkerName = workerName });
        return response;
    }
```

- Para API obtener más información, consulte GetActivityTaskla AWS SDK for .NET APIReferencia.

`ListActivities`

En el siguiente ejemplo de código, se muestra cómo usar `ListActivities`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List the Step Functions activities for the current account.
    /// </summary>
    /// <returns>A list of ActivityListItems.</returns>
    public async Task<List<ActivityListItem>> ListActivitiesAsync()
    {
        var request = new ListActivitiesRequest();
```

```
        var activities = new List<ActivityListItem>();

        do
        {
            var response = await _amazonStepFunctions.ListActivitiesAsync(request);

            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }

            activities.AddRange(response.Activities);
        }
        while (request.NextToken is not null);

        return activities;
    }
```

- Para API obtener más información, consulte ListActivitiesla AWS SDK for .NET APIReferencia.

`ListExecutions`

En el siguiente ejemplo de código, se muestra cómo usar `ListExecutions`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Retrieve information about executions of a Step Functions
    /// state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>A list of ExecutionListItem objects.</returns>
```

```
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }
```

- Para API obtener más información, consulte ListExecutionsla AWS SDK for .NET APIReferencia.

`ListStateMachines`

En el siguiente ejemplo de código, se muestra cómo usar `ListStateMachines`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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 Step Functions state machines.
    /// </summary>
```

```
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
 ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }
```

- Para API obtener más información, consulte ListStateMachinesla AWS SDK for .NET APIReferencia.

`SendTaskSuccess`

En el siguiente ejemplo de código, se muestra cómo usar `SendTaskSuccess`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
    /// <param name="taskResponse">The response received from executing the task.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte SendTaskSuccessla AWS SDK for .NET APIReferencia.

`StartExecution`

En el siguiente ejemplo de código, se muestra cómo usar `StartExecution`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
    {
        var executionRequest = new StartExecutionRequest
        {
```

```
            Input = executionJson,
            StateMachineArn = stateMachineArn
        };

        var response = await
_amazonStepFunctions.StartExecutionAsync(executionRequest);
        return response.ExecutionArn;
    }
```

- Para API obtener más información, consulte StartExecutionla AWS SDK for .NET APIReferencia.

AWS STS ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with AWS STS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones

Acciones

`AssumeRole`

En el siguiente ejemplo de código, se muestra cómo usar `AssumeRole`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

namespace AssumeRoleExample
{
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";
```

```
            var client = new
 Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
 user.
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arn}");

            // Create the request to use with the AssumeRoleAsync call.
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
 assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
 assumeRoleRes.Credentials);

            // Get and display information about the caller that has assumed the
 defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
        }
    }
}
```

- Para API obtener más información, consulte AssumeRolela AWS SDK for .NET APIReferencia.

AWS Support ejemplos que utilizan AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes mediante el uso del AWS SDK for .NET with AWS Support.

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 funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Introducción

¿Hola AWS Support

En los siguientes ejemplos de código se muestra cómo empezar a utilizar AWS Glue.

AWS SDK for .NET

> **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.AWSSupport;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public static class HelloSupport
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the AWS Support service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        // You must have one of the following AWS Support plans: Business,
 Enterprise On-Ramp, or Enterprise. Otherwise, an exception will be thrown.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonAWSSupport>()
            ).Build();

        // Now the client is available for injection.
        var supportClient = host.Services.GetRequiredService<IAmazonAWSSupport>();
```

```
        // You can use await and any of the async methods to get a response.
        var response = await supportClient.DescribeServicesAsync();
        Console.WriteLine($"\tHello AWS Support! There are {response.Services.Count}
 services available.");
    }
}
```

- Para API obtener más información, consulte DescribeServicesla AWS SDK for .NET APIReferencia.

Temas

- Conceptos básicos
- Acciones

Conceptos básicos

Aprenda los conceptos básicos

En el siguiente ejemplo de código, se muestra cómo:

- Obtenga y muestre los servicios disponibles y los niveles de gravedad de los casos.
- Cree un caso de asistencia mediante un servicio, una categoría y un nivel de gravedad seleccionados.
- Obtenga y muestre una lista de casos abiertos para el día actual.
- Añada una serie de archivos adjuntos y una comunicación al nuevo caso.
- Describa el nuevo archivo adjunto y la comunicación del caso.
- Resuelva el caso.
- Obtenga y muestre una lista de casos resueltos para el día actual.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre 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.

```
/// <summary>
/// Hello AWS Support example.
/// </summary>
public static class SupportCaseScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    To use the AWS Support API, you must have one of the following AWS Support
 plans: Business, Enterprise On-Ramp, or Enterprise.

    This .NET example performs the following tasks:
    1.  Get and display services. Select a service from the list.
    2.  Select a category from the selected service.
    3.  Get and display severity levels and select a severity level from the list.
    4.  Create a support case using the selected service, category, and severity
 level.
    5.  Get and display a list of open support cases for the current day.
    6.  Create an attachment set with a sample text file to add to the case.
    7.  Add a communication with the attachment to the support case.
    8.  List the communications of the support case.
    9.  Describe the attachment set.
    10. Resolve the support case.
    11. Get a list of resolved cases for the current day.
   */

    private static SupportWrapper _supportWrapper = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS Support service.
        // Use your AWS profile name, or leave it blank to use the default profile.
```

```
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonAWSSupport>(new AWSOptions() { Profile
= "default" })
                    .AddTransient<SupportWrapper>()
            )
            .Build();

        var logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger(typeof(SupportCaseScenario));

        _supportWrapper = host.Services.GetRequiredService<SupportWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS Support case example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var apiSupported = await _supportWrapper.VerifySubscription();
            if (!apiSupported)
            {
                logger.LogError("You must have a Business, Enterprise On-Ramp, or
Enterprise Support " +
                                "plan to use the AWS Support API. \n\tPlease
upgrade your subscription to run these examples.");
                return;
            }

            var service = await DisplayAndSelectServices();

            var category = DisplayAndSelectCategories(service);

            var severityLevel = await DisplayAndSelectSeverity();

            var caseId = await CreateSupportCase(service, category, severityLevel);
```

```
            await DescribeTodayOpenCases();

            var attachmentSetId = await CreateAttachmentSet();

            await AddCommunicationToCase(attachmentSetId, caseId);

            var attachmentId = await ListCommunicationsForCase(caseId);

            await DescribeCaseAttachment(attachmentId);

            await ResolveCase(caseId);

            await DescribeTodayResolvedCases();

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("AWS Support case example scenario complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// List some available services from AWS Support, and select a service for the
example.
    /// </summary>
    /// <returns>The selected service.</returns>
    private static async Task<Service> DisplayAndSelectServices()
    {
        Console.WriteLine(new string('-', 80));
        var services = await _supportWrapper.DescribeServices();
        Console.WriteLine($"AWS Support client returned {services.Count}
services.");

        Console.WriteLine($"1. Displaying first 10 services:");
        for (int i = 0; i < 10 && i < services.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {services[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > services.Count)
```

```
        {
            Console.WriteLine(
                "Select an example support service by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        Console.WriteLine(new string('-', 80));

        return services[choiceNumber - 1];
    }

    /// <summary>
    /// List the available categories for a service and select a category for the
example.
    /// </summary>
    /// <param name="service">Service to use for displaying categories.</param>
    /// <returns>The selected category.</returns>
    private static Category DisplayAndSelectCategories(Service service)
    {
        Console.WriteLine(new string('-', 80));

        Console.WriteLine($"2. Available support categories for Service
\"{service.Name}\":");
        for (int i = 0; i < service.Categories.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {service.Categories[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > service.Categories.Count)
        {
            Console.WriteLine(
                "Select an example support category by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        Console.WriteLine(new string('-', 80));

        return service.Categories[choiceNumber - 1];
    }
```

```
    /// <summary>
    /// List available severity levels from AWS Support, and select a level for the
example.
    /// </summary>
    /// <returns>The selected severity level.</returns>
    private static async Task<SeverityLevel> DisplayAndSelectSeverity()
    {
        Console.WriteLine(new string('-', 80));
        var severityLevels = await _supportWrapper.DescribeSeverityLevels();

        Console.WriteLine($"3. Get and display available severity levels:");
        for (int i = 0; i < 10 && i < severityLevels.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {severityLevels[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > severityLevels.Count)
        {
            Console.WriteLine(
                "Select an example severity level by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        Console.WriteLine(new string('-', 80));

        return severityLevels[choiceNumber - 1];
    }

    /// <summary>
    /// Create an example support case.
    /// </summary>
    /// <param name="service">Service to use for the new case.</param>
    /// <param name="category">Category to use for the new case.</param>
    /// <param name="severity">Severity to use for the new case.</param>
    /// <returns>The caseId of the new support case.</returns>
    private static async Task<string> CreateSupportCase(Service service,
        Category category, SeverityLevel severity)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Create an example support case" +
                          $" with the following settings:" +
```

```
                        $" \n\tService: {service.Name}, Category: {category.Name}
" +
                        $"and Severity Level: {severity.Name}.");
        var caseId = await _supportWrapper.CreateCase(service.Code, category.Code,
severity.Code,
            "Example case for testing, ignore.", "This is my example support
case.");

        Console.WriteLine($"\tNew case created with ID {caseId}");

        Console.WriteLine(new string('-', 80));

        return caseId;
    }

    /// <summary>
    /// List open cases for the current day.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeTodayOpenCases()
    {
        Console.WriteLine($"5. List the open support cases for the current day.");
        // Describe the cases. If it is empty, try again and allow time for the new
case to appear.
        List<CaseDetails> currentOpenCases = null!;
        while (currentOpenCases == null || currentOpenCases.Count == 0)
        {
            Thread.Sleep(1000);
            currentOpenCases = await _supportWrapper.DescribeCases(
                new List<string>(),
                null,
                false,
                false,
                DateTime.UtcNow.Date,
                DateTime.UtcNow);
        }

        foreach (var openCase in currentOpenCases)
        {
            Console.WriteLine($"\tCase: {openCase.CaseId} created
{openCase.TimeCreated}");
        }

        Console.WriteLine(new string('-', 80));
```

```
    }

    /// <summary>
    /// Create an attachment set for a support case.
    /// </summary>
    /// <returns>The attachment set id.</returns>
    private static async Task<string> CreateAttachmentSet()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Create an attachment set for a support case.");
        var fileName = "example_attachment.txt";

        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for attachment to a support case.");
        }

        await using var ms = new MemoryStream(await
 File.ReadAllBytesAsync(fileName));

        var attachmentSetId = await _supportWrapper.AddAttachmentToSet(
            ms,
            fileName);

        Console.WriteLine($"\tNew attachment set created with id: \n
\t{attachmentSetId.Substring(0, 65)}...");

        Console.WriteLine(new string('-', 80));

        return attachmentSetId;
    }

    /// <summary>
    /// Add an attachment set and communication to a case.
    /// </summary>
    /// <param name="attachmentSetId">Id of the attachment set.</param>
    /// <param name="caseId">Id of the case to receive the attachment set.</param>
    /// <returns>Async task.</returns>
    private static async Task AddCommunicationToCase(string attachmentSetId, string
 caseId)
    {
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Add attachment set and communication to {caseId}.");

        await _supportWrapper.AddCommunicationToCase(
            caseId,
            "This is an example communication added to a support case.",
            attachmentSetId);

        Console.WriteLine($"\tNew attachment set and communication added to
{caseId}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the communications for a case.
    /// </summary>
    /// <param name="caseId">Id of the case to describe.</param>
    /// <returns>An attachment id.</returns>
    private static async Task<string> ListCommunicationsForCase(string caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. List communications for case {caseId}.");

        var communications = await _supportWrapper.DescribeCommunications(caseId);
        var attachmentId = "";
        foreach (var communication in communications)
        {
            Console.WriteLine(
                $"\tCommunication created on: {communication.TimeCreated} has
{communication.AttachmentSet.Count} attachments.");
            if (communication.AttachmentSet.Any())
            {
                attachmentId = communication.AttachmentSet.First().AttachmentId;
            }
        }

        Console.WriteLine(new string('-', 80));
        return attachmentId;
    }

    /// <summary>
    /// Describe an attachment by id.
    /// </summary>
```

```
    /// <param name="attachmentId">Id of the attachment to describe.</param>
    /// <returns>Async task.</returns>
    private static async Task DescribeCaseAttachment(string attachmentId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Describe the attachment set.");

        var attachment = await _supportWrapper.DescribeAttachment(attachmentId);
        var data = Encoding.ASCII.GetString(attachment.Data.ToArray());
        Console.WriteLine($"\tAttachment includes {attachment.FileName} with data:
\n\t{data}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Resolve the support case.
    /// </summary>
    /// <param name="caseId">Id of the case to resolve.</param>
    /// <returns>Async task.</returns>
    private static async Task ResolveCase(string caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Resolve case {caseId}.");

        var status = await _supportWrapper.ResolveCase(caseId);
        Console.WriteLine($"\tCase {caseId} has final status {status}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List resolved cases for the current day.
    /// </summary>
    /// <returns>Async Task.</returns>
    private static async Task DescribeTodayResolvedCases()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. List the resolved support cases for the current
day.");
        var currentCases = await _supportWrapper.DescribeCases(
            new List<string>(),
            null,
            false,
```

```
            true,
            DateTime.UtcNow.Date,
            DateTime.UtcNow);

        foreach (var currentCase in currentCases)
        {
            if (currentCase.Status == "resolved")
            {
                Console.WriteLine(
                    $"\tCase: {currentCase.CaseId}: status {currentCase.Status}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }
}
```

Métodos envoltorios utilizados por el escenario para AWS Support las acciones.

```
/// <summary>
/// Wrapper methods to use AWS Support for working with support cases.
/// </summary>
public class SupportWrapper
{
    private readonly IAmazonAWSSupport _amazonSupport;
    public SupportWrapper(IAmazonAWSSupport amazonSupport)
    {
        _amazonSupport = amazonSupport;
    }


    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
 are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(
```

```
            new DescribeServicesRequest()
            {
                Language = language
            });
        return response.Services;
    }



    /// <summary>
    /// Get the descriptions of support severity levels.
    /// </summary>
    /// <param name="name">Optional language for severity levels.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <returns>The list of support severity levels.</returns>
    public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
    {
        var response = await _amazonSupport.DescribeSeverityLevelsAsync(
            new DescribeSeverityLevelsRequest()
            {
                Language = language
            });
        return response.SeverityLevels;
    }



    /// <summary>
    /// Create a new support case.
    /// </summary>
    /// <param name="serviceCode">Service code for the new case.</param>
    /// <param name="categoryCode">Category for the new case.</param>
    /// <param name="severityCode">Severity code for the new case.</param>
    /// <param name="subject">Subject of the new case.</param>
    /// <param name="body">Body text of the new case.</param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
    /// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
```

```
   /// <returns>The caseId of the new support case.</returns>
   public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
       string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
   {
       var response = await _amazonSupport.CreateCaseAsync(
           new CreateCaseRequest()
           {
               ServiceCode = serviceCode,
               CategoryCode = categoryCode,
               SeverityCode = severityCode,
               Subject = subject,
               Language = language,
               AttachmentSetId = attachmentSetId,
               IssueType = issueType,
               CommunicationBody = body
           });
       return response.CaseId;
   }



   /// <summary>
   /// Add an attachment to a set, or create a new attachment set if one does not
exist.
   /// </summary>
   /// <param name="data">The data for the attachment.</param>
   /// <param name="fileName">The file name for the attachment.</param>
   /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
   /// <returns>The setId of the attachment.</returns>
   public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
   {
       var response = await _amazonSupport.AddAttachmentsToSetAsync(
           new AddAttachmentsToSetRequest
           {
               AttachmentSetId = attachmentSetId,
               Attachments = new List<Attachment>
               {
                   new Attachment
                   {
                       Data = data,
```

```
                        FileName = fileName
                    }
                }
            });
        return response.AttachmentSetId;
    }



    /// <summary>
    /// Get description of a specific attachment.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
    /// <returns>The attachment object.</returns>
    public async Task<Attachment> DescribeAttachment(string attachmentId)
    {
        var response = await _amazonSupport.DescribeAttachmentAsync(
            new DescribeAttachmentRequest()
            {
                AttachmentId = attachmentId
            });
        return response.Attachment;
    }



    /// <summary>
    /// Add communication to a case, including optional attachment set ID and CC
email addresses.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
    /// <param name="body">Body text of the communication.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set.</param>
    /// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> AddCommunicationToCase(string caseId, string body,
        string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
    {
        var response = await _amazonSupport.AddCommunicationToCaseAsync(
            new AddCommunicationToCaseRequest()
            {
                CaseId = caseId,
                CommunicationBody = body,
```

```
                AttachmentSetId = attachmentSetId,
                CcEmailAddresses = ccEmailAddresses
            });
        return response.Result;
    }



    /// <summary>
    /// Describe the communications for a case, optionally with a date filter.
    /// </summary>
    /// <param name="caseId">The ID of the support case.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <returns>The list of communications for the case.</returns>
    public async Task<List<Communication>> DescribeCommunications(string caseId,
 DateTime? afterTime = null, DateTime? beforeTime = null)
    {
        var results = new List<Communication>();
        var paginateCommunications =
 _amazonSupport.Paginators.DescribeCommunications(
            new DescribeCommunicationsRequest()
            {
                CaseId = caseId,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s")
            });
        // Get the entire list using the paginator.
        await foreach (var communications in paginateCommunications.Communications)
        {
            results.Add(communications);
        }
        return results;
    }



    /// <summary>
    /// Get case details for a list of case ids, optionally with date filters.
    /// </summary>
    /// <param name="caseIds">The list of case IDs.</param>
    /// <param name="displayId">Optional display ID.</param>
```

```
    /// <param name="includeCommunication">True to include communication. Defaults
 to true.</param>
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
 to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
 are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
 displayId = null, bool includeCommunication = true,
        bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
 beforeTime = null,
        string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }



    /// <summary>
    /// Resolve a support case by caseId.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
```

```
    /// <returns>The final status of the case after resolving.</returns>
    public async Task<string> ResolveCase(string caseId)
    {
        var response = await _amazonSupport.ResolveCaseAsync(
            new ResolveCaseRequest()
            {
                CaseId = caseId
            });
        return response.FinalCaseStatus;
    }


    /// <summary>
    /// Verify the support level for AWS Support API access.
    /// </summary>
    /// <returns>True if the subscription level supports API access.</returns>
    public async Task<bool> VerifySubscription()
    {
        try
        {
            var response = await _amazonSupport.DescribeServicesAsync(
                new DescribeServicesRequest()
                {
                    Language = "en"
                });
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Amazon.AWSSupport.AmazonAWSSupportException ex)
        {
            if (ex.ErrorCode == "SubscriptionRequiredException")
            {
                return false;
            }
            else throw;
        }
    }
}
```

- Para API obtener más información, consulte los siguientes temas en AWS SDK for .NET APIReference.
  - AddAttachmentsToSet

- AddCommunicationToCase
- CreateCase
- DescribeAttachment
- DescribeCases
- DescribeCommunications
- DescribeServices
- DescribeSeverityLevels
- ResolveCase

Acciones

`AddAttachmentsToSet`

En el siguiente ejemplo de código, se muestra cómo usar `AddAttachmentsToSet`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Add an attachment to a set, or create a new attachment set if one does not
exist.
   /// </summary>
   /// <param name="data">The data for the attachment.</param>
   /// <param name="fileName">The file name for the attachment.</param>
   /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
   /// <returns>The setId of the attachment.</returns>
   public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
   {
       var response = await _amazonSupport.AddAttachmentsToSetAsync(
           new AddAttachmentsToSetRequest
```

```
            {
                AttachmentSetId = attachmentSetId,
                Attachments = new List<Attachment>
                {
                    new Attachment
                    {
                        Data = data,
                        FileName = fileName
                    }
                }
            });
        return response.AttachmentSetId;
    }
```

- Para API obtener más información, consulte AddAttachmentsToSetla AWS SDK for .NET APIReferencia.

`AddCommunicationToCase`

En el siguiente ejemplo de código, se muestra cómo usar `AddCommunicationToCase`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Add communication to a case, including optional attachment set ID and CC
email addresses.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
    /// <param name="body">Body text of the communication.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set.</param>
    /// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> AddCommunicationToCase(string caseId, string body,
        string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
    {
        var response = await _amazonSupport.AddCommunicationToCaseAsync(
            new AddCommunicationToCaseRequest()
            {
                CaseId = caseId,
                CommunicationBody = body,
                AttachmentSetId = attachmentSetId,
                CcEmailAddresses = ccEmailAddresses
            });
        return response.Result;
    }
```

- Para API obtener más información, consulte AddCommunicationToCasela AWS SDK for .NET APIReferencia.

`CreateCase`

En el siguiente ejemplo de código, se muestra cómo usar `CreateCase`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Create a new support case.
    /// </summary>
    /// <param name="serviceCode">Service code for the new case.</param>
    /// <param name="categoryCode">Category for the new case.</param>
    /// <param name="severityCode">Severity code for the new case.</param>
    /// <param name="subject">Subject of the new case.</param>
    /// <param name="body">Body text of the new case.</param>
    /// <param name="language">Optional language support for your case.
```

```
   /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
   /// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
   /// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
   /// <returns>The caseId of the new support case.</returns>
   public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
       string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
   {
       var response = await _amazonSupport.CreateCaseAsync(
           new CreateCaseRequest()
           {
               ServiceCode = serviceCode,
               CategoryCode = categoryCode,
               SeverityCode = severityCode,
               Subject = subject,
               Language = language,
               AttachmentSetId = attachmentSetId,
               IssueType = issueType,
               CommunicationBody = body
           });
       return response.CaseId;
   }
```

- Para API obtener más información, consulte CreateCasela AWS SDK for .NET APIReferencia.

`DescribeAttachment`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeAttachment`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get description of a specific attachment.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
    /// <returns>The attachment object.</returns>
    public async Task<Attachment> DescribeAttachment(string attachmentId)
    {
        var response = await _amazonSupport.DescribeAttachmentAsync(
            new DescribeAttachmentRequest()
            {
                AttachmentId = attachmentId
            });
        return response.Attachment;
    }
```

- Para API obtener más información, consulte DescribeAttachmentla AWS SDK for .NET APIReferencia.

`DescribeCases`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeCases`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get case details for a list of case ids, optionally with date filters.
    /// </summary>
    /// <param name="caseIds">The list of case IDs.</param>
    /// <param name="displayId">Optional display ID.</param>
```

```
    /// <param name="includeCommunication">True to include communication. Defaults
 to true.</param>
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
 to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
 are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
 displayId = null, bool includeCommunication = true,
        bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
 beforeTime = null,
        string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }
```

- Para API obtener más información, consulte DescribeCasesla AWS SDK for .NET APIReferencia.

`DescribeCommunications`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeCommunications`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Describe the communications for a case, optionally with a date filter.
    /// </summary>
    /// <param name="caseId">The ID of the support case.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <returns>The list of communications for the case.</returns>
    public async Task<List<Communication>> DescribeCommunications(string caseId,
 DateTime? afterTime = null, DateTime? beforeTime = null)
    {
        var results = new List<Communication>();
        var paginateCommunications =
 _amazonSupport.Paginators.DescribeCommunications(
            new DescribeCommunicationsRequest()
            {
                CaseId = caseId,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s")
            });
        // Get the entire list using the paginator.
        await foreach (var communications in paginateCommunications.Communications)
        {
            results.Add(communications);
        }
        return results;
    }
```

- Para API obtener más información, consulte DescribeCommunicationsla AWS SDK for .NET APIReferencia.

`DescribeServices`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeServices`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = language
            });
        return response.Services;
    }
```

- Para API obtener más información, consulte DescribeServicesla AWS SDK for .NET APIReferencia.

`DescribeSeverityLevels`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeSeverityLevels`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Get the descriptions of support severity levels.
   /// </summary>
   /// <param name="name">Optional language for severity levels.
   /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
   /// <returns>The list of support severity levels.</returns>
   public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
   {
       var response = await _amazonSupport.DescribeSeverityLevelsAsync(
           new DescribeSeverityLevelsRequest()
           {
               Language = language
           });
       return response.SeverityLevels;
   }
```

- Para API obtener más información, consulte DescribeSeverityLevelsla AWS SDK for .NET APIReferencia.

`ResolveCase`

En el siguiente ejemplo de código, se muestra cómo usar `ResolveCase`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Resolve a support case by caseId.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}
```

- Para API obtener más información, consulte ResolveCasela AWS SDK for .NET APIReferencia.

Ejemplos de Amazon Textract usando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET con Amazon Textract.

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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Escenarios

Creación de una aplicación para analizar los comentarios de los clientes

El siguiente ejemplo de código muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para ver el código fuente y las instrucciones de implementación, consulta el proyecto en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ejemplos de Amazon Transcribe utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET con Amazon Transcribe.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

Acciones

`CreateVocabulary`

En el siguiente ejemplo de código, se muestra cómo usar `CreateVocabulary`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Create a custom vocabulary using a list of phrases. Custom vocabularies
   /// improve transcription accuracy for one or more specific words.
   /// </summary>
   /// <param name="languageCode">The language code of the vocabulary.</param>
   /// <param name="phrases">Phrases to use in the vocabulary.</param>
   /// <param name="vocabularyName">Name for the vocabulary.</param>
   /// <returns>The state of the custom vocabulary.</returns>
   public async Task<VocabularyState> CreateCustomVocabulary(LanguageCode
languageCode,
       List<string> phrases, string vocabularyName)
   {
       var response = await _amazonTranscribeService.CreateVocabularyAsync(
           new CreateVocabularyRequest
           {
               LanguageCode = languageCode,
```

```
                Phrases = phrases,
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- Para API obtener más información, consulte CreateVocabularyla AWS SDK for .NET APIReferencia.

`DeleteMedicalTranscriptionJob`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteMedicalTranscriptionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a medical transcription job. Also deletes the transcript associated
 with the job.
    /// </summary>
    /// <param name="jobName">Name of the medical transcription job to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMedicalTranscriptionJob(string jobName)
    {
        var response = await
 _amazonTranscribeService.DeleteMedicalTranscriptionJobAsync(
            new DeleteMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteMedicalTranscriptionJobla AWS SDK for .NET APIReferencia.

`DeleteTranscriptionJob`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteTranscriptionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Delete a transcription job. Also deletes the transcript associated with the
job.
    /// </summary>
    /// <param name="jobName">Name of the transcription job to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTranscriptionJob(string jobName)
    {
        var response = await _amazonTranscribeService.DeleteTranscriptionJobAsync(
            new DeleteTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para API obtener más información, consulte DeleteTranscriptionJobla AWS SDK for .NET APIReferencia.

`DeleteVocabulary`

En el siguiente ejemplo de código, se muestra cómo usar `DeleteVocabulary`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Delete an existing custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.DeleteVocabularyAsync(
        new DeleteVocabularyRequest
        {
            VocabularyName = vocabularyName
        });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para API obtener más información, consulte DeleteVocabularyla AWS SDK for .NET APIReferencia.

`GetTranscriptionJob`

En el siguiente ejemplo de código, se muestra cómo usar `GetTranscriptionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get details about a transcription job.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <returns>A TranscriptionJob instance with information on the requested
job.</returns>
    public async Task<TranscriptionJob> GetTranscriptionJob(string jobName)
    {
        var response = await _amazonTranscribeService.GetTranscriptionJobAsync(
            new GetTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.TranscriptionJob;
    }
```

- Para API obtener más información, consulte GetTranscriptionJobla AWS SDK for .NET APIReferencia.

`GetVocabulary`

En el siguiente ejemplo de código, se muestra cómo usar `GetVocabulary`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Get information about a custom vocabulary.
    /// </summary>
    /// <param name="vocabularyName">Name of the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> GetCustomVocabulary(string vocabularyName)
    {
        var response = await _amazonTranscribeService.GetVocabularyAsync(
            new GetVocabularyRequest()
            {
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- Para API obtener más información, consulte GetVocabularyla AWS SDK for .NET APIReferencia.

`ListMedicalTranscriptionJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListMedicalTranscriptionJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List medical transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the medical
transcription jobs.</param>
    /// <returns>A list of summaries about medical transcription jobs.</returns>
```

```
    public async Task<List<MedicalTranscriptionJobSummary>>
ListMedicalTranscriptionJobs(
        string? jobNameContains = null)
    {
        var response = await
_amazonTranscribeService.ListMedicalTranscriptionJobsAsync(
            new ListMedicalTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.MedicalTranscriptionJobSummaries;
    }
```

- Para API obtener más información, consulte ListMedicalTranscriptionJobsla AWS SDK for .NET APIReferencia.

`ListTranscriptionJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListTranscriptionJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the transcription
jobs.</param>
    /// <returns>A list of transcription job summaries.</returns>
    public async Task<List<TranscriptionJobSummary>> ListTranscriptionJobs(string?
jobNameContains = null)
    {
        var response = await _amazonTranscribeService.ListTranscriptionJobsAsync(
```

```
            new ListTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.TranscriptionJobSummaries;
    }
```

- Para API obtener más información, consulte ListTranscriptionJobsla AWS SDK for .NET APIReferencia.

`ListVocabularies`

En el siguiente ejemplo de código, se muestra cómo usar `ListVocabularies`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// List custom vocabularies for the current account. Optionally specify a name
    /// filter and a specific state to filter the vocabularies list.
    /// </summary>
    /// <param name="nameContains">Optional string the vocabulary name must
contain.</param>
    /// <param name="stateEquals">Optional state of the vocabulary.</param>
    /// <returns>List of information about the vocabularies.</returns>
    public async Task<List<VocabularyInfo>> ListCustomVocabularies(string?
nameContains = null,
        VocabularyState? stateEquals = null)
    {
        var response = await _amazonTranscribeService.ListVocabulariesAsync(
            new ListVocabulariesRequest()
            {
                NameContains = nameContains,
```

```
                StateEquals = stateEquals
            });
        return response.Vocabularies;
    }
```

- Para API obtener más información, consulte ListVocabulariesla AWS SDK for .NET APIReferencia.

`StartMedicalTranscriptionJob`

En el siguiente ejemplo de código, se muestra cómo usar `StartMedicalTranscriptionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Start a medical transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the medical transcription job.</
param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
 location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
    /// <param name="outputBucketName">Location for the output, typically an Amazon
 S3 location.</param>
    /// <param name="transcriptionType">Conversation or dictation transcription
 type.</param>
    /// <returns>A MedicalTransactionJob instance with information on the new job.</
returns>
    public async Task<MedicalTranscriptionJob> StartMedicalTranscriptionJob(
        string jobName, string mediaFileUri,
```

```
        MediaFormat mediaFormat, string outputBucketName,
Amazon.TranscribeService.Type transcriptionType)
    {
        var response = await
_amazonTranscribeService.StartMedicalTranscriptionJobAsync(
            new StartMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
                LanguageCode =
                    LanguageCode
                        .EnUS, // The value must be en-US for medical
transcriptions.
                OutputBucketName = outputBucketName,
                OutputKey =
                    jobName, // The value is a key used to fetch the output of the
transcription.
                Specialty = Specialty.PRIMARYCARE, // The value PRIMARYCARE must be
set.
                Type = transcriptionType
            });
        return response.MedicalTranscriptionJob;
    }
```

- Para API obtener más información, consulte StartMedicalTranscriptionJobla AWS SDK for .NET APIReferencia.

`StartTranscriptionJob`

En el siguiente ejemplo de código, se muestra cómo usar `StartTranscriptionJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Start a transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
 location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
    /// <param name="languageCode">The language code of the media file, such as en-
US.</param>
    /// <param name="vocabularyName">Optional name of a custom vocabulary.</param>
    /// <returns>A TranscriptionJob instance with information on the new job.</
returns>
    public async Task<TranscriptionJob> StartTranscriptionJob(string jobName, string
 mediaFileUri,
        MediaFormat mediaFormat, LanguageCode languageCode, string? vocabularyName)
    {
        var response = await _amazonTranscribeService.StartTranscriptionJobAsync(
            new StartTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
                LanguageCode = languageCode,
                Settings = vocabularyName != null ? new Settings()
                {
                    VocabularyName = vocabularyName
                } : null
            });
        return response.TranscriptionJob;
```

```
    }
```

- Para API obtener más información, consulte StartTranscriptionJobla AWS SDK for .NET APIReferencia.

`UpdateVocabulary`

En el siguiente ejemplo de código, se muestra cómo usar `UpdateVocabulary`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   /// <summary>
   /// Update a custom vocabulary with new values. Update overwrites all existing
information.
   /// </summary>
   /// <param name="languageCode">The language code of the vocabulary.</param>
   /// <param name="phrases">Phrases to use in the vocabulary.</param>
   /// <param name="vocabularyName">Name for the vocabulary.</param>
   /// <returns>The state of the custom vocabulary.</returns>
   public async Task<VocabularyState> UpdateCustomVocabulary(LanguageCode
languageCode,
       List<string> phrases, string vocabularyName)
   {
       var response = await _amazonTranscribeService.UpdateVocabularyAsync(
           new UpdateVocabularyRequest()
           {
               LanguageCode = languageCode,
               Phrases = phrases,
               VocabularyName = vocabularyName
           });
       return response.VocabularyState;
   }
```

- Para API obtener más información, consulte UpdateVocabularyla AWS SDK for .NET APIReferencia.

Ejemplos de Amazon Translate utilizando AWS SDK for .NET

Los siguientes ejemplos de código muestran cómo realizar acciones e implementar escenarios comunes AWS SDK for .NET mediante Amazon Translate.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, 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.

Cada ejemplo incluye un enlace al código fuente completo, donde puede encontrar instrucciones sobre cómo configurar y ejecutar el código en su contexto.

Temas

- Acciones
- Escenarios

Acciones

`DescribeTextTranslationJob`

En el siguiente ejemplo de código, se muestra cómo usar `DescribeTextTranslationJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

    /// <summary>
    /// The following example shows how to retrieve the details of
    /// a text translation job using Amazon Translate.
    /// </summary>
    public class DescribeTextTranslation
    {
        public static async Task Main()
        {
            var client = new AmazonTranslateClient();

            // The Job Id is generated when the text translation job is started
            // with a call to the StartTextTranslationJob method.
            var jobId = "1234567890abcdef01234567890abcde";

            var request = new DescribeTextTranslationJobRequest
            {
                JobId = jobId,
            };

            var jobProperties = await DescribeTranslationJobAsync(client, request);

            DisplayTranslationJobDetails(jobProperties);
        }

        /// <summary>
        /// Retrieve information about an Amazon Translate text translation job.
        /// </summary>
        /// <param name="client">The initialized Amazon Translate client object.</
param>
        /// <param name="request">The DescribeTextTranslationJobRequest object.</
param>
        /// <returns>The TextTranslationJobProperties object containing
        /// information about the text translation job..</returns>
        public static async Task<TextTranslationJobProperties>
 DescribeTranslationJobAsync(
            AmazonTranslateClient client,
            DescribeTextTranslationJobRequest request)
        {
```

```
            var response = await client.DescribeTextTranslationJobAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                return response.TextTranslationJobProperties;
            }
            else
            {
                return null;
            }
        }

        /// <summary>
        /// Displays the properties of the text translation job.
        /// </summary>
        /// <param name="jobProperties">The properties of the text translation
        /// job returned by the call to DescribeTextTranslationJobAsync.</param>
        public static void DisplayTranslationJobDetails(TextTranslationJobProperties
jobProperties)
        {
            if (jobProperties is null)
            {
                Console.WriteLine("No text translation job properties found.");
                return;
            }

            // Display the details of the text translation job.
            Console.WriteLine($"{jobProperties.JobId}: {jobProperties.JobName}");
        }
    }
```

- Para API obtener más información, consulte DescribeTextTranslationJobla AWS SDK for .NET APIReferencia.

`ListTextTranslationJobs`

En el siguiente ejemplo de código, se muestra cómo usar `ListTextTranslationJobs`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Translate;
    using Amazon.Translate.Model;

    /// <summary>
    /// List Amazon Translate translation jobs, along with details about each job.
    /// </summary>
    public class ListTranslationJobs
    {
        public static async Task Main()
        {
            var client = new AmazonTranslateClient();
            var filter = new TextTranslationJobFilter
            {
                JobStatus = "COMPLETED",
            };

            var request = new ListTextTranslationJobsRequest
            {
                MaxResults = 10,
                Filter = filter,
            };

            await ListJobsAsync(client, request);
        }

        /// <summary>
        /// List Amazon Translate text translation jobs.
        /// </summary>
        /// <param name="client">The initialized Amazon Translate client object.</
param>
        /// <param name="request">An Amazon Translate
```

```
        /// ListTextTranslationJobsRequest object detailing which text
        /// translation jobs are of interest.</param>
        public static async Task ListJobsAsync(
            AmazonTranslateClient client,
            ListTextTranslationJobsRequest request)
        {
            ListTextTranslationJobsResponse response;

            do
            {
                response = await client.ListTextTranslationJobsAsync(request);

ShowTranslationJobDetails(response.TextTranslationJobPropertiesList);

                request.NextToken = response.NextToken;
            }
            while (response.NextToken is not null);
        }

        /// <summary>
        /// List existing translation job details.
        /// </summary>
        /// <param name="properties">A list of Amazon Translate text
        /// translation jobs.</param>
        public static void
ShowTranslationJobDetails(List<TextTranslationJobProperties> properties)
        {
            properties.ForEach(prop =>
            {
                Console.WriteLine($"{prop.JobId}: {prop.JobName}");
                Console.WriteLine($"Status: {prop.JobStatus}");
                Console.WriteLine($"Submitted time: {prop.SubmittedTime}");
            });
        }
    }
```

- Para API obtener más información, consulte ListTextTranslationJobsla AWS SDK for .NET APIReferencia.

`StartTextTranslationJob`

En el siguiente ejemplo de código, se muestra cómo usar `StartTextTranslationJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Translate;
    using Amazon.Translate.Model;

    /// <summary>
    /// This example shows how to use Amazon Translate to process the files in
    /// an Amazon Simple Storage Service (Amazon S3) bucket. The translated results
    /// will also be stored in an Amazon S3 bucket.
    /// </summary>
    public class BatchTranslate
    {
        public static async Task Main()
        {
            var contentType = "text/plain";

            // Set this variable to an S3 bucket location with a folder."
            // Input files must be in a folder and not at the bucket root."
            var s3InputUri = "s3://DOC-EXAMPLE-BUCKET1/FOLDER/";
            var s3OutputUri = "s3://DOC-EXAMPLE-BUCKET2/";

            // This role must have permissions to read the source bucket and to read
 and
            // write to the destination bucket where the translated text will be
 stored.
            var dataAccessRoleArn = "arn:aws:iam::0123456789ab:role/
S3TranslateRole";

            var client = new AmazonTranslateClient();
```

```
            var inputConfig = new InputDataConfig
            {
                ContentType = contentType,
                S3Uri = s3InputUri,
            };

            var outputConfig = new OutputDataConfig
            {
                S3Uri = s3OutputUri,
            };

            var request = new StartTextTranslationJobRequest
            {
                JobName = "ExampleTranslationJob",
                DataAccessRoleArn = dataAccessRoleArn,
                InputDataConfig = inputConfig,
                OutputDataConfig = outputConfig,
                SourceLanguageCode = "en",
                TargetLanguageCodes = new List<string> { "fr" },
            };

            var response = await StartTextTranslationAsync(client, request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"{response.JobId}: {response.JobStatus}");
            }
        }

        /// <summary>
        /// Start the Amazon Translate text translation job.
        /// </summary>
        /// <param name="client">The initialized AmazonTranslateClient object.</
param>
        /// <param name="request">The request object that includes details such
        /// as source and destination bucket names and the IAM Role that will
        /// be used to access the buckets.</param>
        /// <returns>The StartTextTranslationResponse object that includes the
        /// details of the request response.</returns>
        public static async Task<StartTextTranslationJobResponse>
 StartTextTranslationAsync(AmazonTranslateClient client,
 StartTextTranslationJobRequest request)
        {
            var response = await client.StartTextTranslationJobAsync(request);
```

```
            return response;
        }
    }
```

- Para API obtener más información, consulte StartTextTranslationJobla AWS SDK for .NET APIReferencia.

`StopTextTranslationJob`

En el siguiente ejemplo de código, se muestra cómo usar `StopTextTranslationJob`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

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

/// <summary>
/// Shows how to stop a running Amazon Translation Service text translation
/// job.
/// </summary>
public class StopTextTranslationJob
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new StopTextTranslationJobRequest
        {
            JobId = jobId,
        };
```

```
            await StopTranslationJobAsync(client, request);
        }

        /// <summary>
        /// Sends a request to stop a text translation job.
        /// </summary>
        /// <param name="client">Initialized AmazonTrnslateClient object.</param>
        /// <param name="request">The request object to be passed to the
        /// StopTextJobAsync method.</param>
        public static async Task StopTranslationJobAsync(
            AmazonTranslateClient client,
            StopTextTranslationJobRequest request)
        {
            var response = await client.StopTextTranslationJobAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"{response.JobId} as status:
{response.JobStatus}");
            }
        }
    }
```

- Para API obtener más información, consulte StopTextTranslationJobla AWS SDK for .NET APIReferencia.

`TranslateText`

En el siguiente ejemplo de código, se muestra cómo usar `TranslateText`.

AWS SDK for .NET

> **Note**
>
> Hay más información sobre GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    using System;
```

```
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Transfer;
    using Amazon.Translate;
    using Amazon.Translate.Model;

    /// <summary>
    /// Take text from a file stored a Amazon Simple Storage Service (Amazon S3)
    /// object and translate it using the Amazon Transfer Service.
    /// </summary>
    public class TranslateText
    {
        public static async Task Main()
        {
            // If the region you want to use is different from the region
            // defined for the default user, supply it as a parameter to the
            // Amazon Translate client object constructor.
            var client = new AmazonTranslateClient();

            // Set the source language to "auto" to request Amazon Translate to
            // automatically detect te language of the source text.

            // You can get a list of the languages supposed by Amazon Translate
            // in the Amazon Translate Developer's Guide here:
            //      https://docs.aws.amazon.com/translate/latest/dg/what-is.html
            string srcLang = "en"; // English.
            string destLang = "fr"; // French.

            // The Amazon Simple Storage Service (Amazon S3) bucket where the
            // source text file is stored.
            string srcBucket = "DOC-EXAMPLE-BUCKET";
            string srcTextFile = "source.txt";

            var srcText = await GetSourceTextAsync(srcBucket, srcTextFile);
            var destText = await TranslatingTextAsync(client, srcLang, destLang,
srcText);

            ShowText(srcText, destText);
        }

        /// <summary>
        /// Use the Amazon S3 TransferUtility to retrieve the text to translate
        /// from an object in an S3 bucket.
```

```
        /// </summary>
        /// <param name="srcBucket">The name of the S3 bucket where the
        /// text is stored.
        /// </param>
        /// <param name="srcTextFile">The key of the S3 object that
        /// contains the text to translate.</param>
        /// <returns>A string representing the source text.</returns>
        public static async Task<string> GetSourceTextAsync(string srcBucket, string
srcTextFile)
        {
            string srcText = string.Empty;

            var s3Client = new AmazonS3Client();
            TransferUtility utility = new TransferUtility(s3Client);

            using var stream = await utility.OpenStreamAsync(srcBucket,
srcTextFile);

            StreamReader file = new System.IO.StreamReader(stream);

            srcText = file.ReadToEnd();
            return srcText;
        }

        /// <summary>
        /// Use the Amazon Translate Service to translate the document from the
        /// source language to the specified destination language.
        /// </summary>
        /// <param name="client">The Amazon Translate Service client used to
        /// perform the translation.</param>
        /// <param name="srcLang">The language of the source text.</param>
        /// <param name="destLang">The destination language for the translated
        /// text.</param>
        /// <param name="text">A string representing the text to ranslate.</param>
        /// <returns>The text that has been translated to the destination
        /// language.</returns>
        public static async Task<string> TranslatingTextAsync(AmazonTranslateClient
client, string srcLang, string destLang, string text)
        {
            var request = new TranslateTextRequest
            {
                SourceLanguageCode = srcLang,
                TargetLanguageCode = destLang,
                Text = text,
```

```
            };

            var response = await client.TranslateTextAsync(request);

            return response.TranslatedText;
        }

        /// <summary>
        /// Show the original text followed by the translated text.
        /// </summary>
        /// <param name="srcText">The original text to be translated.</param>
        /// <param name="destText">The translated text.</param>
        public static void ShowText(string srcText, string destText)
        {
            Console.WriteLine("Source text:");
            Console.WriteLine(srcText);
            Console.WriteLine();
            Console.WriteLine("Translated text:");
            Console.WriteLine(destText);
        }
    }
```

- Para API obtener más información, consulte TranslateTextla AWS SDK for .NET APIReferencia.

Escenarios

Creación de una SNS aplicación de Amazon

El siguiente ejemplo de código muestra cómo crear una aplicación que tenga funciones de suscripción y publicación y que traduzca los mensajes.

AWS SDK for .NET

Muestra cómo utilizar el Amazon Simple Notification Service. NETAPIpara crear una aplicación web con funciones de suscripción y publicación. Además, esta aplicación de ejemplo también traduce los mensajes.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarla y ejecutarla, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon SNS
- Amazon Translate

Creación de una aplicación para analizar los comentarios de los clientes

El siguiente ejemplo de código muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Seguridad de 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

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. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es 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 la sección Privacidad de datos FAQ. Para obtener información sobre la protección de datos en Europa, consulte el modelo de responsabilidad AWS compartida y la entrada del GDPR blog sobre AWS seguridad.

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:

- Utilice la autenticación multifactorial (MFA) con cada cuenta.
- UseSSL/TLSpara comunicarse con AWS los recursos. Necesitamos TLS 1.2 y recomendamos TLS 1.3.
- Configure API y registre la actividad del usuario con AWS CloudTrail.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita entre FIPS 140 y 3 módulos criptográficos validados para acceder a AWS través de una interfaz de línea de comandos o unaAPI, utilice un FIPS terminal. Para obtener más información sobre los FIPS puntos finales disponibles, consulte la Norma federal de procesamiento de información () 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, API AWS CLI, o. AWS SDKs Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, le recomendamos

encarecidamente que no incluya información sobre las credenciales URL para validar su solicitud a ese servidor.

Identity and Access Management

AWS Identity and Access Management (IAM) es una servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a AWS los recursos. IAMlos administradores controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar AWS los recursos. IAMes una servicio de AWS que puede utilizar sin coste adicional.

Temas

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 AWS de 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. A continuación, debe enviar solicitudes a su IAM administrador para cambiar los permisos de los usuarios del servicio. Revise la información de esta página para comprender los conceptos básicos deIAM. Para obtener más información sobre cómo puede usarlo IAM su empresa AWS, consulte la guía del usuario del servicio de AWS que está utilizando.

IAMadministrador: si es IAM administrador, tal vez le interese obtener más información sobre cómo puede redactar políticas para administrar el acceso AWS. Para ver ejemplos de políticas AWS basadas en la identidad que puede utilizarIAM, 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 estar autenticado (con quien haya iniciado sesión AWS) como IAM usuario o asumiendo un IAM rol. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (IAMIdentity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, el administrador configuró previamente la federación de identidades mediante roles. IAM Cuando accede AWS mediante la federación, asume indirectamente un rol.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS incluye un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar AWS API las solicitudes en la Guía del IAM usuario.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactorial (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactorial en la Guía del AWS IAM Identity Center usuario y Uso de la autenticación multifactorial (MFA) AWS en la Guía del IAM usuario.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el

usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de tareas que requieren que inicie sesión como usuario root, consulte Tareas que requieren credenciales de usuario root en la Guía del IAM usuario.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, 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 de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en IAM Identity Center, o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus aplicaciones Cuentas de AWS . Para obtener información sobre IAM Identity Center, consulte ¿Qué es IAM Identity Center? en la Guía AWS IAM Identity Center del usuario.

Usuarios y grupos de IAM

Un IAMusuario es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos utilizar credenciales temporales en lugar de crear IAM usuarios con credenciales de larga duración, como contraseñas y claves de acceso. Sin embargo, si tiene casos de uso específicos que requieren credenciales a largo plazo con IAM los usuarios, le recomendamos que rote las claves de acceso. Para obtener más información, consulte Rotar las claves de acceso con regularidad para los casos de uso que requieran credenciales de larga duración en la Guía del IAM usuario.

Un IAMgrupo es una identidad que especifica un conjunto de IAM usuarios. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos para grandes conjuntos de usuarios. Por ejemplo, puede asignar un nombre a un grupo IAMAdminsy concederle permisos para administrar IAM los recursos.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte Cuándo crear un IAM usuario (en lugar de un rol) en la Guía del IAM usuario.

IAMroles

Un IAMrol es una identidad dentro de ti Cuenta de AWS que tiene permisos específicos. Es similar a un IAM usuario, pero no está asociado a una persona específica. Puede asumir temporalmente un IAM rol en el AWS Management Console cambiando de rol. Puede asumir un rol llamando a una AWS API operación AWS CLI o o utilizando una operación personalizadaURL. Para obtener más información sobre los métodos de uso de roles, consulte Uso de IAM roles en la Guía del IAM usuario.

IAMlos roles con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información sobre los roles para la federación, consulte Creación de un rol para un proveedor de identidad externo en la Guía del IAM usuario. Si usa IAM Identity Center, configura un conjunto de permisos. Para controlar a qué pueden acceder sus identidades después de autenticarse, IAM Identity Center correlaciona el conjunto de permisos con un rol en. IAM Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .
- Permisos IAM de usuario temporales: un IAM usuario o rol puede asumir un IAM rol para asumir temporalmente diferentes permisos para una tarea específica.
- Acceso multicuenta: puedes usar un IAM rol para permitir que alguien (un responsable de confianza) de una cuenta diferente acceda a los recursos de tu cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunos servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para conocer la diferencia entre las funciones y las políticas basadas en recursos para el acceso multicuenta, consulta el tema sobre el acceso a los recursos entre cuentas IAM en la Guía del IAM usuario.
- Acceso entre servicios: algunos servicios de AWS utilizan funciones en otros. servicios de AWS Por ejemplo, cuando realizas una llamada en un servicio, es habitual que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio

haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

- Sesiones de acceso directo (FAS): cuando utilizas un IAM usuario o un rol para realizar acciones en AWS ellas, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FASutiliza los permisos del principal que llama a an servicio de AWS, junto con los que solicitan, servicio de AWS para realizar solicitudes a los servicios descendentes. FASlas solicitudes solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros recursos servicios de AWS o para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información detallada sobre la política a la hora de realizar FAS solicitudes, consulte Reenviar las sesiones de acceso.
- Función de servicio: una función de servicio es una IAMfunción que un servicio asume para realizar acciones en su nombre. Un IAM administrador puede crear, modificar y eliminar un rol de servicio desde dentroIAM. Para obtener más información, consulte Crear un rol para delegar permisos servicio de AWS en un rol en el IAMManual del usuario.
- Función vinculada a un servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un IAM administrador puede ver los permisos de los roles vinculados al servicio, pero no editarlos.

- Aplicaciones que se ejecutan en Amazon EC2: puedes usar un IAM rol para administrar las credenciales temporales de las aplicaciones que se ejecutan en una EC2 instancia y que realizan AWS CLI o AWS API solicitan. Esto es preferible a almacenar las claves de acceso dentro de la EC2 instancia. Para asignar un AWS rol a una EC2 instancia y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite que los programas que se ejecutan en la EC2 instancia obtengan credenciales temporales. Para obtener más información, consulte Uso de un IAM rol para conceder permisos a aplicaciones que se ejecutan en EC2 instancias de Amazon en la Guía del IAM usuario.

Para saber si se deben usar IAM roles o IAM usuarios, consulte Cuándo crear un IAM rol (en lugar de un usuario) en la Guía del IAM usuario.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos.

AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como JSON documentos. Para obtener más información sobre la estructura y el contenido de los documentos de JSON políticas, consulte Descripción general de JSON las políticas en la Guía del IAM usuario.

Los administradores pueden usar AWS JSON las políticas para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder a los usuarios permiso para realizar acciones en los recursos que necesitan, un IAM administrador puede crear IAM políticas. A continuación, el administrador puede añadir las IAM políticas a las funciones y los usuarios pueden asumir las funciones.

IAMlas políticas definen los permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de AWS Management Console AWS CLI, el o el AWS API.

Políticas basadas en identidad

Las políticas basadas en la identidad son documentos de política de JSON permisos que se pueden adjuntar a una identidad, como un IAM usuario, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener información sobre cómo crear una política basada en la identidad, consulte Creación de IAM políticas en la Guía del usuario. IAM

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y funciones de su empresa. Cuenta de AWS Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para saber cómo elegir entre una política gestionada o una política integrada, consulte Elegir entre políticas gestionadas y políticas integradas en la Guía del IAM usuario.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de JSON política que se adjuntan a un recurso. Algunos ejemplos de políticas basadas en recursos son las políticas de confianza de IAM roles 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. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puede usar políticas AWS administradas desde una política IAM 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. ACLsson similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de JSON políticas.

Amazon S3 AWS WAF y Amazon VPC son ejemplos de servicios compatiblesACLs. Para obtener más informaciónACLs, 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 y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una función avanzada en la que se establecen los permisos máximos que una política basada en la identidad puede conceder a una IAM entidad (IAMusuario o rol). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte los límites de los permisos para IAM las entidades en la Guía del IAMusuario.
- Políticas de control de servicios (SCPs): SCPs son JSON políticas que especifican los permisos máximos para una organización o unidad organizativa (OU) en AWS Organizations. AWS Organizations es un servicio para agrupar y administrar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilitas todas las funciones de una organización, puedes aplicar políticas de control de servicios (SCPs) a una o a todas tus cuentas. SCPLimita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada

una Usuario raíz de la cuenta de AWS. Para obtener más información sobre OrganizationsSCPs, consulte las políticas de control de servicios en la Guía del AWS Organizations usuario.

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información, consulte las políticas de sesión en la Guía del IAM usuario.

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 IAM usuario.

¿Cómo servicios de AWS trabajar con IAM

Para obtener una visión general de cómo servicios de AWS funciona con la mayoría de las IAM funciones, consulte AWS los servicios con los que funcionan IAM en la Guía del IAM usuario.

Para obtener información sobre cómo usar una función específica servicio de AWS IAM, consulta la sección de seguridad de la Guía del usuario del servicio correspondiente.

Solución de problemas AWS de identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas más comunes que pueden surgir al trabajar con AWS yIAM.

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.

El siguiente ejemplo de error se produce cuando el `mateojackson` IAM usuario intenta usar la consola para ver los detalles de un *`my-example-widget`* recurso ficticio, pero no tiene los `awes:`*`GetWidget`* permisos ficticios.

```
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 AWS el 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.

Algunas 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 el rol al servicio.

El siguiente ejemplo de error se produce cuando un IAM usuario denominado `marymajor` intenta utilizar la consola para realizar una acción en ella. 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 AWS el 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

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. 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 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 propiedad, consulte Proporcionar acceso a un IAM usuario en otro Cuenta de AWS de su propiedad en la Guía del IAM usuario. Cuentas de AWS
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo permitir el acceso a recursos que Cuentas de AWS son propiedad de terceros en la Guía del IAM usuario.
- Para obtener información sobre cómo proporcionar acceso mediante la federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (federación de identidades) en la Guía del IAM usuario.
- Para saber la diferencia entre el uso de roles y políticas basadas en recursos para el acceso entre cuentas, consulte el acceso a recursos entre cuentas IAM en la Guía del usuario. 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. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Guías de inicio rápido sobre seguridad y cumplimiento: estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en AWS la seguridad y el cumplimiento.

- Diseñando una arquitectura basada en la HIPAA seguridad y el cumplimiento en Amazon Web Services: en este documento técnico se describe cómo pueden utilizar las empresas AWS para crear HIPAA aplicaciones aptas.

> **Note**
>
> No todos son aptos. servicios de AWS HIPAA Para obtener más información, consulta la Referencia de servicios HIPAA aptos.

- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- AWS Guías de cumplimiento para clientes: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. En las guías se resumen las mejores prácticas para garantizar la seguridad servicios de AWS y se orientan a los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad de la Industria de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- Evaluación de los recursos con reglas en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- AWS Security Hub— Esto servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la Referencia de controles de Security Hub.
- Amazon GuardDuty: servicio de AWS detecta posibles amenazas para sus cargas de trabajo Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, por ejemplo PCIDSS, cumpliendo con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.
- AWS Audit Manager— Esto le servicio de AWS ayuda a auditar continuamente su AWS consumo para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

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 API llamadas 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). Necesitamos TLS 1.2 y recomendamos TLS 1.3.

- Cifre suites con perfecto secreto (PFS), como (Ephemeral Diffie-Hellman) o DHE 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 firmarse con un identificador de clave de acceso y una clave de acceso secreta que esté asociada a un director. IAM También puede 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 TLS versión mínima en el AWS SDK for .NET

Para aumentar la seguridad al comunicarse con AWS los servicios, debe AWS SDK for .NET configurar la versión TLS 1.2 o una versión posterior.

AWS SDK for .NET Utiliza lo subyacente. NETtiempo de ejecución para determinar qué protocolo de seguridad utilizar. De forma predeterminada, las versiones actuales de. NETutilice el último protocolo configurado compatible con el sistema operativo. La aplicación puede anular este SDK comportamiento, pero no se recomienda hacerlo.

. NETNúcleo

Por defecto,. NETCore usa el último protocolo configurado compatible con el sistema operativo. El AWS SDK for .NET no proporciona un mecanismo para anular esto.

Si está utilizando un. NETEn la versión principal anterior a la 2.1, le recomendamos encarecidamente que actualice su. NETVersión básica.

Consulte lo siguiente para obtener información específica de cada sistema operativo.

Windows

Las distribuciones modernas de Windows tienen habilitada la compatibilidad con la versión TLS 1.2 de forma predeterminada. Si utiliza Windows 7 SP1 o Windows Server 2008 R2SP1, debe

asegurarse de que la compatibilidad con la versión TLS 1.2 esté habilitada en el registro, tal y como se describe en https://learn.microsoft.com/en-us/tls-registry-settingswindows-server/security/tls/ #tls-12. Si está ejecutando una distribución anterior, debe actualizar el sistema operativo. Para obtener información sobre la compatibilidad con la versión TLS 1.3 en Windows, consulte la documentación más reciente de Microsoft para conocer las versiones de cliente o servidor mínimas requeridas.

macOS

Si estás corriendo. NETCore 2.1 o posterior, TLS 1.2 está habilitada de forma predeterminada. TLSLa versión 1.2 es compatible con OS X Mavericks v10.9 o posterior. . NETLa versión principal 2.1 y las posteriores requieren versiones más recientes de macOS, como se describe en https://learn.microsoft.com/en-us/dotnet/core/install/windows? tabs=net80&pivots=os-macos.

Si estás usando. NETCore 1.0,. NETCore usa Open SSL en macOS, una dependencia que debe instalarse por separado. Open SSL agregó soporte para TLS 1.2 en la versión 1.0.1 y agregó soporte para TLS 1.3 en la versión 1.1.1.

Linux

. NETCore para Linux requiere OpenSSL, que viene incluido en muchas distribuciones de Linux. Pero también se puede instalar por separado. Open SSL agregó soporte para TLS 1.2 en la versión 1.0.1 y agregó soporte para TLS 1.3 en la versión 1.1.1. Si utilizas una versión moderna de. NETCore (2.1 o posterior) y has instalado un administrador de paquetes, es probable que tengas instalada una versión más moderna de OpenSSL.

Para estar seguro, puede ejecutar **`openssl version`** en un terminal y verificar que la versión sea posterior a 1.0.1.

. NETMarco

Si está ejecutando una versión moderna de. NETFramework (4.7 o posterior) y una versión moderna de Windows (al menos Windows 8 para los clientes y Windows Server 2012 o posterior para los servidores), la versión TLS 1.2 está habilitada y se usa de forma predeterminada.

Si utilizas un. NETTiempo de ejecución de Framework que no usa la configuración del sistema operativo (. NETFramework 3.5 a 4.5.2), AWS SDK for .NET intentará añadir soporte para TLS 1.1 y TLS 1.2 a los protocolos compatibles. Si estás usando. NETFramework 3.5, esto solo tendrá éxito si se instala el parche activo adecuado, de la siguiente manera:

- Windows 10 versión 1511 y Windows Server 2016: KB3156421

- Windows 8.1 y Windows Server 2012 R2 — 0 KB315452
- Windows Server 2012 — KB3154519
- Windows 7 SP1 y Server 2008 R2 SP1 — KB3154518

> **Warning**
>
> A partir del 15 de agosto de 2024, AWS SDK for .NET finalizará el soporte para. NETFramework 3.5 y cambiará el mínimo. NETVersión Framework a la 4.7.2. Para obtener más información, consulte la entrada del blog Se avecinan cambios importantes. NETObjetivos de los marcos 3.5 y 4.5 del AWS SDK for .NET.

Si su aplicación se ejecuta en una versión más nueva. NETEn Windows 7 SP1 o Windows Server 2008 R2SP1, debe asegurarse de que la compatibilidad con la versión TLS 1.2 esté habilitada en el registro, tal y como se describe en https://learn.microsoft.com/en-us/tls-registry-settingswindows-server/security/tls/ #tls -12. Las versiones más recientes de Windows lo tienen habilitado de forma predeterminada.

Para obtener información detallada sobre las prácticas recomendadas para su uso con. TLS NETFramework, consulte el artículo de Microsoft en https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls.

AWS Tools for PowerShell

AWS Tools for PowerShellutilice AWS SDK for .NET el para AWS todas las llamadas a los servicios. El comportamiento del entorno depende de la versión de Windows PowerShell que esté ejecutando, de la siguiente manera.

De Windows PowerShell 2.0 a 5.x

Se ejecuta en Windows PowerShell 2.0 a 5.x. NETMarco. Puedes verificar cuál. NETEl tiempo de ejecución (2.0 o 4.0) se utiliza PowerShell mediante el siguiente comando.

```
$PSVersionTable.CLRVersion
```

- Cuando se usa. NETRuntime 2.0, siga las instrucciones proporcionadas anteriormente con respecto a AWS SDK for .NET y. NETFramework 3.5.

> **Warning**
>
> A partir del 15 de agosto de 2024, AWS SDK for .NET finalizará el soporte para. NETFramework 3.5 y cambiará el mínimo. NETVersión Framework a la 4.7.2. Para obtener más información, consulte la entrada del blog Se avecinan cambios importantes. NETObjetivos de los marcos 3.5 y 4.5 del AWS SDK for .NET.

- Cuando se usa. NETRuntime 4.0, siga las instrucciones proporcionadas anteriormente con respecto a AWS SDK for .NET y. NETFramework 4+.

Windows 6.0 PowerShell

Se ejecuta Windows PowerShell 6.0 y versiones posteriores. NETNúcleo. Puede comprobar qué versión de. NETPara utilizar Core, ejecute el siguiente comando.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Siga las instrucciones proporcionadas anteriormente con respecto AWS SDK for .NET a la versión correspondiente de. NETNúcleo.

Xamarin

Para Xamarin, consulta las instrucciones en xamarin/cross-platform/app-fundamentals/https://learn.microsoft.com/en-us/. transport-layer-security En resumen:

Para Android

- Requiere Android 5.0 o posterior.
- Propiedades del proyecto , opciones de Android: la implementación debe estar configurada en Android y la HttpClient implementación/debe estar configurada en Native 1.2+. SSL TLS TLS

Para iOS

- Requiere iOS 7 o posterior.
- Propiedades del proyecto, compilación de iOS: HttpClient la implementación debe estar configurada en NSUrlSession.

Para macOS

- Requiere macOS 10.9 o posterior.
- Opciones de proyecto, compilación, compilación para Mac: HttpClient la implementación debe configurarse en NSUrlSession.

Unity

Debe usar Unity 2018.2 o una versión posterior y usar el. NET4.x Tiempo de ejecución de secuencias de comandos equivalente. 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/ScriptingRuntimeUpgradeDocumentation/Manual/ .html. El. NETEl tiempo de ejecución de secuencias de comandos equivalente a 4.x permite la compatibilidad con la versión TLS 1.2 con todas las plataformas Unity que ejecutan Mono o. IL2CPP

Navegador (para Blazor) WebAssembly

WebAssembly se ejecuta en el navegador en lugar de en el servidor y utiliza el navegador para gestionar el HTTP tráfico. Por lo tanto, TLS el soporte viene determinado por el soporte del navegador.

Blazor WebAssembly, en versión preliminar deASP. NETCore 3.1 solo es compatible con navegadores compatibles, tal y como se describe en WebAssembly https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms. Todos los WebAssembly navegadores convencionales eran TLS compatibles con la versión 1.2 antes de ser compatibles. Si este es el caso de tu navegador, si tu aplicación se ejecuta, podrá comunicarse a través de la versión TLS 1.2.

Consulte la documentación de su navegador para obtener más información y verificación.

Migración de clientes de cifrado de Amazon S3

En este tema se muestra cómo migrar las aplicaciones de la versión 1 (V1) del cliente de cifrado de Amazon Simple Storage Service (Amazon 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 API compatible con el cliente de la versión 1, por lo que no es necesario realizar ningún otro cambio en el 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 Amazon.Extensions.S3.Encryption. NuGet 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. KMSCuando se usa:

        i. Proporcione un identificador KMS clave.

        ii. Declare el método de cifrado que está utilizando, esto es, `KmsType.KmsContext`.

        iii. Proporcione un contexto de cifrado 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: KMS a KMS +Context

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 (de AES - CBC a AES - GCM Key Wrap)

`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 hacer para verificar que el objeto no se ha modificado desde que se cifró.

Ejemplo: algoritmo asimétrico (RSApara RSA - OAEP - SHA1 Key Wrap)

`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);
```

Consideraciones especiales de AWS SDK for .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 for .NET

En este tema se describe cómo obtener los AWSSDK ensamblados 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 Instalación de paquetes de AWSSDK 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 for .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-net45.zip: ensamblajes compatibles con .NET Framework 4.5 y versiones posteriores.
- aws-sdk-net35.zip: ensamblajes compatibles con .NET Framework 3.5.

> **Warning**
>
> A partir del 15 de agosto de 2024, AWS SDK for .NET dejarán de ser compatibles con .NET Framework 3.5 y cambiarán 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 3.5 y 4.5 de .NET Framework. AWS SDK for .NET

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 Instalación de ensamblados de AWSSDK sin NuGet.

Acceso a las credenciales y perfiles en una aplicación

El método preferido para usar credenciales es dejar que AWS SDK for .NET las encuentre y recupere en su nombre, tal y como se describe en Resolución de credencial y perfil.

Con todo, también puede configurar la aplicación para recuperar perfiles y credenciales de forma activa y luego usar explícitamente dichas credenciales al crear un cliente de servicio de AWS.

Para recuperar perfiles y credenciales de forma activa, utilice las clases del espacio de nombres Amazon.Runtime.CredentialManagement.

- Para encontrar un perfil en un archivo que usa el formato de archivo de credenciales de AWS (ya sea el archivo de credenciales de AWS compartido en su ubicación predeterminada o un archivo de credenciales personalizado), utilice la clase SharedCredentialsFile. En aras de una mayor brevedad, en este texto los archivos con este formato a veces se denominarán simplemente archivos de credenciales.
- Para encontrar un perfil en SDK Store, use la clase NetSDKCredentialsFile.

- Para buscar tanto en un archivo de credenciales como en SDK Store, según la configuración de una propiedad de clase, use la clase CredentialProfileStoreChain.

  Esta clase se puede usar para encontrar perfiles. También se puede usar para solicitar credenciales de AWS directamente en lugar de usar la clase `AWSCredentialsFactory` (se describe a continuación).
- Para recuperar o crear varios tipos de credenciales de un perfil, utilice la clase AWSCredentialsFactory.

En las siguientes secciones se muestran ejemplos de estas clases.

Ejemplos de la clase CredentialProfileStoreChain

Se pueden obtener credenciales o perfiles de la clase CredentialProfileStoreChain con los métodos TryGetAWSCredentials o TryGetProfile. La propiedad `ProfilesLocation` de la clase determina el comportamiento de los métodos del siguiente modo:

- Si `ProfilesLocation` es nula o está vacía, busque en SDK Store si la plataforma la admite y, a continuación, busque el archivo de credenciales de AWS compartido en la ubicación predeterminada.
- Si la propiedad `ProfilesLocation` contiene un valor, busque en el archivo de credenciales especificado en la propiedad.

Obtención de credenciales desde SDK Store o un archivo de credenciales de AWS compartido

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 la clase Amazon.Runtime.AWSCredentials.

```
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}");
    }
}
```

Obtención de un perfil desde SDK Store o un archivo de credenciales de AWS compartido

En este ejemplo se muestra cómo obtener un perfil mediante la clase CredentialProfileStoreChain. 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 la clase CredentialProfile.

```
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 clase CredentialProfileStoreChain. Las credenciales proceden de un archivo que usa el formato de archivo de credenciales de AWS, pero que se encuentra en otra ubicación.

En este ejemplo también se usa la clase Amazon.Runtime.AWSCredentials.

```
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 las clases SharedCredentialsFile y AWSCredentialsFactory

Creación de un objeto AmazonS3Client mediante la clase SharedCredentialsFile

En este ejemplo se muestra cómo encontrar un perfil en el archivo de credenciales de AWS compartido, cómo crear credenciales de AWS desde el perfil y, a continuación, cómo utilizar esas credenciales para crear un objeto AmazonS3Client. En el ejemplo se usa la clase SharedCredentialsFile.

En este ejemplo se usan también las clases CredentialProfile y Amazon.Runtime.AWSCredentials.

```
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 NetSdkCredentialsFile se puede usar exactamente de la misma manera, salvo que se crearía una instancia de un nuevo objeto NetSdkCredentialsFile y no de un objeto SharedCredentialsFile.

Consideraciones especiales sobre la compatibilidad con Unity

Si utiliza AWS SDK for .NET y .NET Standard 2.0 en una aplicación de Unity, esta debe hacer referencia directa a los ensamblados de AWS SDK for .NET (archivos DLL) en vez de usar NuGet. Debido a este requisito, estas son algunas acciones importantes que deberá realizar.

- Debe obtener los ensamblados de AWS SDK for .NET y aplicarlos al proyecto. Para obtener información sobre cómo hacerlo, consulte Descarga y extracción de archivos ZIP en el tema Obtención de AWSSDK conjuntos.

- Debe incluir los siguientes archivos DLL en el proyecto de Unity junto con los archivos DLL de AWSSDK.Core y de los demás servicios de AWS que utilice. A partir de la versión 3.5.109 de AWS SDK for .NET, el archivo ZIP de .NET Standard contiene estos archivos DLL adicionales.
  - Microsoft.Bcl.AsyncInterfaces.dll
  - System.Runtime.CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- Si utiliza IL2CPP para crear el proyecto de Unity, debe agregar un archivo `link.xml` a la carpeta Asset para evitar la extracción de código. El archivo `link.xml` debe contener todos los ensamblados de AWSSDK que use, y cada uno debe incluir el atributo `preserve="all"`. 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-from-unity-xamarin-or-uwp/.

Además de estas consideraciones especiales, consulte ¿Qué ha cambiado en la versión 3.5? para obtener información sobre cómo migrar la aplicación de Unity a la versión 3.5 de AWS SDK for .NET.

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 de AWS SDK for .NET

AWS SDK for .NET proporciona una API que se puede usar para acceder a servicios de AWS. Para ver qué clases y métodos están disponibles en la API, consulte la Referencia de API de AWS SDK for .NET.

Además de la referencia general indicada anteriormente, cada uno de los ejemplos de la sección Ejemplos de código con orientaciones contiene referencias a los métodos y clases específicos que se utilizan en ese ejemplo.

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 for .NET . Para recibir notificaciones sobre las actualizaciones de esta documentación, puedes suscribirte a un RSS feed.

Cambio	Descripción	Fecha
Novedades	Se agregó información sobre la primera versión prelimina r de la AWS SDK for .NET versión 4.	16 de agosto de 2024
Novedades	Información actualizada sobre los próximos cambios en. NETSoporte marco.	20 de junio de 2024
Novedades	Se agregó información sobre la versión preliminar del marco de procesamiento de AWS mensajes para. NET	28 de marzo de 2024
Novedades	Se incluye información sobre el soporte para. NET8.	23 de febrero de 2024
Novedades	Incluye información sobre los próximos cambios en. NETSoporte marco.	18 de febrero de 2024
Obtención de AWSSDK ensamblajes	Se incluye información sobre los ensamblajes que los soportan. NET8 y versiones posteriores.	8 de enero de 2024
AWS Marco de procesamiento de mensajes para. NET	Se incluye información sobre la versión beta de Message Processing Framework.	10 de diciembre de 2023

AWS OpsWorks	Se agregó una nota sobre End of Life for AWS OpsWorks.	8 de diciembre de 2023
Uso de Amazon SQL DynamoDB Sin bases de datos	Información actualizada sobre los modelos de programación de persistencia de documento s y objetos. Ahora se pueden evitar determinadas condicion es de latencia o interbloqueo provocadas por comportam ientos de inicio en frío y de grupo de subprocesos.	15 de noviembre de 2023
Se incluyeron más actualiza ciones de IAM prácticas recomendadas	Guía actualizada para adaptarla a las IAM mejores prácticas. Para obtener más información, consulte las mejores prácticas de seguridad en IAM.	5 de octubre de 2023
Obtención de AWSSDK ensamblajes	Se ha eliminado la informaci ón sobre la instalación AWS SDK for .NET mediante el Herramientas de AWS para Windows instalador (es decir, elMSI), que ha quedado obsoleto.	25 de septiembre de 2023
IAMmejores prácticas, actualizaciones.	Guía actualizada para adaptarla a las IAM mejores prácticas. Para obtener más información, consulte las mejores prácticas de seguridad en IAM.	18 de julio de 2023

Lambda Annotations	Se ha publicado el marco de AWS Lambda anotaciones para su disponibilidad general.	17 de julio de 2023
Novedades	Se ha agregado información sobre la versión preliminar del proveedor de caché distribuida de DynamoDB.	15 de julio de 2023
Índice	El índice se ha actualizado para que los ejemplos de código puedan encontrarse más fácilmente.	8 de junio de 2023
Resolución de la región	Se agregó información sobre cómo se SDK resuelve una especificación de región faltante.	14 de marzo de 2023
Support for the MSI	Se agregó una nota sobre la finalización del soporte para el Herramientas de AWS para Windows instalador.	6 de marzo de 2023
Lambda Annotations (versión preliminar)	Vista previa del marco de AWS Lambda anotaciones.	22 de septiembre de 2022
Implemente aplicaciones en AWS	Se trasladó el contenido principal a un sitio de GitHub Pages: https://aws.github.io/aws-dotnet-deploy/	28 de junio de 2022
RetirarseEC2: clásico	Se agregaron notas sobre la jubilación de -ClassicEC2.	13 de abril de 2022
Inicio de sesión único con AWS SDK for .NET	Se agregó información sobre el inicio de sesión único (SSO) al usar el. AWS SDK for .NET	17 de marzo de 2022

Imponer una versión mínima TLS	Se agregó información sobre la versión TLS 1.3.	16 de marzo de 2022
Trabaja con AWS servicios	Incluye listas de ejemplos de código disponibles en GitHub.	28 de febrero de 2022
Habilitación de SDK métricas	Se ha eliminado la informaci ón sobre la activación de SDK las métricas, que ha quedado obsoleta.	20 de enero de 2022
Implemente aplicaciones en AWS	Se agregó una referencia al AWS Toolkit for Visual Studio, que proporciona una funcional idad de implementación similar a AWS la herramienta de implementación.	26 de octubre de 2021
AWS SDK for .NET guía de consolidación de la versión 3	Las dos guías para desarroll adores de la AWS SDK for .NET versión 3, la «V3" y la «última», se han consolida do en una sola guía bajo la «URLv3".	18 de agosto de 2021
Migración desde. NETEstánd ar 1.3	Support for. NETEl estándar 1.3 del AWS SDK for .NET ha llegado al final de su vida útil.	25 de marzo de 2021
Implemente aplicaciones para AWS (vista previa)	Se agregó información de vista previa sobre la herramien ta de AWS implement ación, que puede usar para implementar una aplicación desde. NETCLI.	15 de marzo de 2021
Versión 3.5 del AWS SDK for .NET	Se AWS SDK for .NET ha publicado la versión 3.5 del.	25 de agosto de 2020

Paginadores	Se han añadido paginadores a muchos clientes de servicios, lo que facilita la paginación de API los resultados.	24 de agosto de 2020
Reintentos y tiempos de espera	Se ha agregado información sobre los modos de reintento.	20 de agosto de 2020
Migración de clientes de cifrado de S3	Se ha agregado información sobre cómo migrar los clientes de cifrado de Amazon S3 de la versión 1 a la versión 2.	7 de agosto de 2020
Uso de KMS claves para el cifrado S3	Se ha actualizado el ejemplo para usar la versión 2 del cliente de cifrado de S3.	6 de agosto de 2020
Migrando desde. NETEstándar 1.3	Se agregó información sobre la finalización del soporte para. NETEstándar 1.3 a finales de 2020.	18 de mayo de 2020
Inicio rápido	Se ha agregado una sección de inicio rápido con configura ción básica y tutoriales para presentar AWS SDK for .NET al lector.	27 de marzo de 2020
Aplicación de la 1.2 TLS	Se agregó información sobre cómo hacer cumplir la versión TLS 1.2 en laSDK.	10 de marzo de 2020

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.