

开发人员指南

AWS SDK for .NET



AWS SDK for .NET: 开发人员指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS SDK for .NET	1
关于本版本	1
SDK 主要版本的维护和支持	1
常见使用案例	2
此部分中的其他主题	2
相关的 AWS 工具	2
适用于 Windows PowerShell 的工具和适用于 PowerShell Core 的工具	2
Toolkit for VS Code	2
Toolkit for Visual Studio	3
Toolkit for Azure DevOps	3
开发工具包和工具参考	3
其他资源	4
开始使用	6
安装和配置工具链	6
跨平台开发	6
在 Windows 上使用 Visual Studio 和 .NET Core 开发	7
后续步骤	7
配置开发工具包身份验证	7
启用和配置 IAM Identity Center	8
将开发工具包配置为使用 IAM Identity Center。	8
开始 AWS 访问门户会话	9
其他信息	10
快速了解	10
简单跨平台应用程序	11
基于 Windows 的简单应用程序	16
后续步骤	21
启动新项目	22
配置 AWS 区域	23
创建具有特定区域的服务客户端	23
为所有服务客户端指定区域	24
区域解析	25
有关中国（北京）区域的特殊信息	26
有关新 AWS 服务的特殊信息	26
通过 NuGet 安装 AWSSDK 程序包	26

从命令行提示符处或终端使用 NuGet	27
从 Visual Studio Solution Explorer 中使用 NuGet	27
从 Package Manager Console 使用 NuGet	28
在没有 NuGet 的情况下安装 AWSSDK 程序集	29
凭证和配置文件解析	30
配置文件解析	31
使用联合用户账户凭证	31
指定角色或临时凭证	32
使用代理凭证	32
用户和角色	32
用户和权限集	33
服务角色	33
高级配置	34
AWSSDK.Extensions.NETCore.Setup 和 IConfiguration	34
配置其他应用程序参数	38
AWS SDK for .NET的配置文件参考	46
使用旧版凭证	57
有关凭证的重要警告和指南	57
使用共享 AWS 凭证文件	58
使用 SDK Store (仅适用于 Windows)	61
软件开发工具包功能	65
异步 API	65
重试和超时	66
重试	67
超时	68
示例	69
分页器	70
我在哪里可以找到分页工具？	70
分页工具能提供什么好处？	70
同步分页与异步分页	70
示例	71
分页工具的其他注意事项	74
其他工具	75
AWS 部署工具	75
AWS Message Processing Framework for .NET	75
高级身份验证	76

单点登录	76
先决条件	77
设置 SSO 配置文件	77
生成和使用 SSO 令牌	78
其他资源	83
教程	83
教程：仅限 .NET 应用程序	83
教程：AWS CLI 和 .NET 应用程序	92
部署到 AWS	100
从 .NET CLI 部署	100
通过 IDE 工具包部署	100
用例	101
ASP.NET Core 应用程序	101
.NET Console 应用程序	102
Blazor 应用程序 WebAssembly	102
AWS Lambda 项目	103
先决条件	103
可用的 Lambda 命令	104
部署步骤	104
迁移项目	106
新增内容	106
支持的平台	107
.NET 内核	107
.NET Standard 2.0	107
.NET Framework 4.5	107
.NET Framework 3.5	108
便携式类库和 Xamarin	108
Unity 支持	108
更多信息	108
迁移到版本 3	109
关于AWS SDK for .NET版本	109
面向开发工具包的架构重新设计	109
重大更改	109
迁移到 3.5 版	111
3.5 版的更改内容	111
迁移同步代码	112

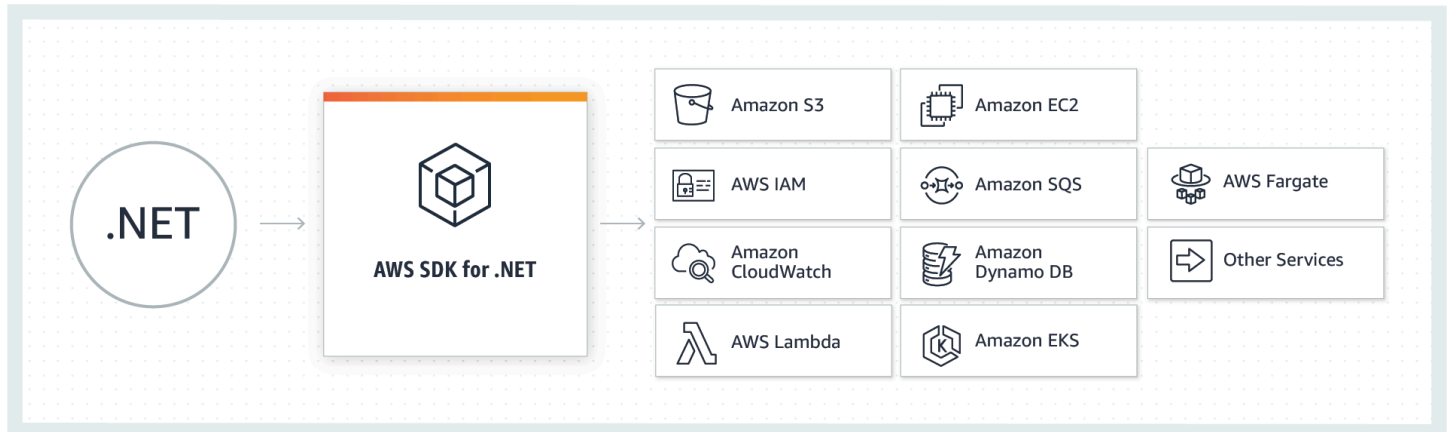
迁移到 3.7 版	113
从 .NET Standard 1.3 迁移	113
使用 AWS 服务	115
带有指导的代码示例	115
AWS CloudFormation	116
Amazon Cognito	118
DynamoDB	126
Amazon EC2	154
IAM	214
Amazon S3	232
Amazon SNS	242
Amazon SQS	246
AWS Lambda	279
API	279
先决条件	279
主题	279
Lambda 注释	279
高级库和框架	281
Message Processing Framework	281
AWS OpsWorks	282
API	282
先决条件	282
其它服务和配置	282
代码示例	284
操作和场景	284
ACM	285
Aurora	290
Auto Scaling	332
Amazon Bedrock	413
Amazon Bedrock 运行时系统	417
CloudWatch	423
CloudWatch 日志	479
Amazon Cognito 身份提供者	493
Amazon Comprehend	518
DynamoDB	529
Amazon EC2	620

Amazon ECS	719
Elastic Load Balancing	732
EventBridge	785
AWS Glue	825
IAM	856
Amazon Keyspaces	981
Kinesis	1009
AWS KMS	1026
Lambda	1038
MediaConvert	1075
企业	1088
Amazon Polly	1106
Amazon RDS	1118
Amazon Rekognition	1152
Route 53 域注册	1182
Amazon S3	1209
S3 Glacier	1304
SageMaker	1314
Secrets Manager	1347
Amazon SES	1350
Amazon SNS	1362
Amazon SQS	1409
Step Functions	1453
AWS STS	1480
AWS Support	1482
Amazon Transcribe	1510
Amazon Translate	1522
跨服务示例	1533
构建 Amazon SNS 应用程序	1533
创建无服务器应用程序来管理照片	1534
创建 Web 应用程序来跟踪 DynamoDB 数据	1534
创建 Aurora Serverless 工作项跟踪器	1535
创建用于分析客户反馈的应用程序	1535
检测图像中的对象	1536
安全性	1537
数据保护	1537

Identity and Access Management	1538
受众	1538
使用身份进行身份验证	1539
使用策略管理访问	1541
AWS 服务如何与 IAM 协同工作	1543
对 AWS 身份和访问进行问题排查	1543
合规性验证	1545
弹性	1546
基础设施安全性	1546
强制实施最低 TLS 版本	1547
.NET 内核	1547
NET Framework。	1548
AWS Tools for PowerShell	1549
Xamarin	1550
Unity	1550
浏览器 (适用于 Blazor WebAssembly)	1550
S3 加密客户端迁移	1551
迁移概述	1551
将现有客户端更新为 V1 过渡客户端以读取新格式	1551
将 V1 过渡客户端迁移到 V2 客户端以写入新格式	1552
将 V2 客户端更新为不再读取 V1 格式	1555
特殊注意事项	1556
获取 AWSSDK 程序集	1556
下载并解压缩 ZIP 文件	1556
访问应用程序中的凭证和配置文件	1557
CredentialProfileStoreChain 类的示例	1557
SharedCredentialsFile 和 AWSCredentialsFactory 类的示例	1559
Unity 支持	1560
Xamarin 支持	1561
API 参考	1562
文档历史记录	1563
.....	mdlxvii

什么是 AWS SDK for .NET

借助AWS SDK for .NET，可以更轻松地构建 .NET 应用程序，应用于经济高效、可扩展而且可靠的 AWS 服务，例如 Amazon Simple Storage Service (Amazon S3) 和 Amazon Elastic Compute Cloud (Amazon EC2)。该软件开发工具包为 .NET 开发人员提供了一组一致且熟悉的库，从而简化了AWS服务的使用。



(好吧，明白了！我已经准备好[快速浏览](#)或开始[设置](#)了。)

关于本版本

Note

本文档仅适用于AWS SDK for .NET版本 3.0 及更高版本。它主要围绕 .NET Core 和 ASP.NET Core，但也包含有关 .NET Framework 和 ASP.NET 4.x 的信息。除了 Windows 和 Visual Studio 之外，它还同样考虑跨平台开发。

有关迁移的信息，请参阅[迁移项目](#)。

要查找早期版本的AWS SDK for .NET的已弃用内容，请参阅以下内容：

- [AWS SDK for .NET \(版本 2, 已弃用 \) 开发人员指南](#)

SDK 主要版本的维护和支持

有关维护和支持 SDK 主要版本及其基础依赖关系的信息，请参阅 [AWS SDK 和工具参考指南](#) 中的以下内容：

- [AWS SDK 和工具维护策略](#)

- [AWS SDK 和工具版本支持矩阵](#)

常见使用案例

AWS SDK for .NET有助于您实现几个引人注目的使用案例，包括以下情况：

- 使用 [AWS Identity and Access Management \(IAM \)](#) 管理用户和角色。
- 访问 [Amazon Simple Storage Service \(Amazon S3 \)](#) 来创建桶和存储对象。
- 管理主题的 [Amazon Simple Notification Service \(Amazon SNS \)](#) HTTP 订阅。
- 使用 [S3 传输实用程序](#)将文件从 Xamarin 应用程序传输到 Amazon S3。
- 使用 [Amazon Simple Queue Service \(Amazon SQS \)](#) 处理系统中组件之间的消息和工作流。
- 通过向 [Amazon S3 Select](#) 发送 SQL 语句，执行高效的 Amazon S3 传输。
- 创建和启动 [Amazon EC2](#) 实例，并配置和请求 Amazon EC2 [竞价型实例](#)。

此部分中的其他主题

- [与AWS SDK for .NET相关的 AWS 工具](#)
- [AWS 开发工具包和工具参考指南](#)
- [其他资源](#)

与AWS SDK for .NET相关的 AWS 工具

适用于 Windows PowerShell 的工具和适用于 PowerShell Core 的工具

AWS Tools for Windows PowerShell 和 AWS Tools for PowerShell Core 是根据AWS SDK for .NET 公开的功能构建的 PowerShell 模块。借助 AWS PowerShell 工具，您可以从 PowerShell 提示符处为针对 AWS 资源的操作编写脚本。虽然 cmdlet 的实现使用了服务客户端和来自开发工具包的方法，cmdlet 提供了惯用的 PowerShell 体验来指定参数和处理结果。

要了解其用法，请参阅 [AWS Tools for Windows PowerShell](#)。

Toolkit for VS Code

[AWS Toolkit for Visual Studio Code](#)是一个适用于 Visual Studio Code (VS 代码) 编辑器的插件。通过该工具包，您可以更轻松地开发、调试和部署使用 AWS 的应用程序。

使用此工具包，您可以执行以下操作：

- 创建包含 AWS Lambda 函数的无服务器应用程序，然后将应用程序部署到 AWS CloudFormation 堆栈。
- 使用 Amazon EventBridge 架构。
- 当处理 Amazon ECS 任务定义文件时使用 IntelliSense。
- 可视化 AWS Cloud Development Kit (AWS CDK) 应用程序。

Toolkit for Visual Studio

AWS Toolkit for Visual Studio 是一个适用于 Visual Studio IDE 的插件，可让您更轻松地进行开发、调试和部署使用 Amazon Web Services 的 .NET 应用程序。Toolkit for Visual Studio 提供 Visual Studio 模板供 Lambda 等服务和部署向导用于 Web 应用程序和无服务器应用程序。您可以使用 AWS Explorer 管理 Amazon EC2 实例、使用 Amazon DynamoDB 表、向 Amazon Simple Notification Service (Amazon SNS) 队列发布消息等，所有这些都可在 Visual Studio 中完成。

要开始使用，请参阅[设置 AWS Toolkit for Visual Studio](#)。

Toolkit for Azure DevOps

AWS Toolkit for Microsoft Azure DevOps 添加了任务，可轻松支持在 Azure DevOps 和 Azure DevOps Server 中构建和发布管道以使用 AWS 服务。您可以使用 Amazon S3、AWS Elastic Beanstalk、AWS CodeDeploy、Lambda、AWS CloudFormation、Amazon Simple Queue Service (Amazon SQS) 和 Amazon SNS。您还可以使用 Windows PowerShell 模块和 AWS Command Line Interface (AWS CLI) 运行命令。

要开始使用 AWS Toolkit for Azure DevOps，请参阅[AWS Toolkit for Microsoft Azure DevOps 用户指南](#)。

AWS 开发工具包和工具参考指南

[AWS 开发工具包和工具参考指南](#)包含与许多 AWS 开发工具包和工具包以及 AWS CLI 相关且重要的信息。以下是参考文献中包含的一些信息示例：

- 有关[共享 AWSconfig 和 credentials 文件](#)及其[位置](#)的信息。
- [设置 AWS 账户、用户和角色](#)
- [配置和身份验证设置参考](#)

- [AWS 通用运行时系统 \(CRT \) 库](#)
- [AWS 开发工具包和工具维护策略](#)
- [AWS 开发工具包和工具版本支持矩阵](#)

其他资源

支持的服务

AWS SDK for .NET支持大多数 AWS 基础设施产品，并且还经常添加更多服务。有关开发工具包支持的 AWS 服务的列表，请参阅[开发工具包自述文件](#)。

修订历史记录

要了解不同版本中发生了哪些变化，请参阅以下内容：

- [开发工具包变更日志](#)
- [AWS SDK for .NET中的新增内容](#)
- [文档历史记录](#)

AWS SDK for .NET主页

有关AWS SDK for .NET的更多信息，请转到位于 <http://aws.amazon.com/sdk-for-net/> 的开发工具包主页。

开发工具包参考文档

开发工具包参考文档提供了浏览和搜索开发工具包中所含全部代码的功能。它提供了详尽的文档和使用示例。有关详细信息，请参阅 [AWS SDK for .NET API 参考](#)。

AWS re:Post (以前是 AWS 论坛)

请访问 [AWS re:Post](#) (特别是[针对AWS SDK for .NET的主题](#))，提出问题或提供 AWS 反馈。每个文档页面的底部都有一个尝试 AWS re:Post 链接，可将您带到相关的 re:Post 主题。AWS 工程师会监控主题并回答疑问、反馈和问题。

如果您已登录 re:Post，也可以关注某个主题。要关注AWS SDK for .NET的主题，请前往[所有主题页面](#)，找到“AWS 上的 .NET”，然后选择关注按钮。

工具包

- [AWS Toolkit for Visual Studio](#) : 如果您使用 Microsoft Visual Studio IDE , 则应该查看 [AWS Toolkit for Visual Studio 用户指南](#)。
- [AWS Toolkit for Visual Studio Code](#) : 如果您使用 Microsoft Visual Studio IDE , 则应该查看 [AWS Toolkit for Visual Studio Code 用户指南](#)。

有用的库、扩展和工具

访问 Github 网站上的 [aws/dotnet](#) 和 [aws/aws-sdk-net](#) 库 , 获得有助于在 AWS 上构建 .NET 应用程序和服务的库、工具和资源的链接。

下面是一些示例 :

- [AWS适用于 Systems Manager 的 .NET 配置扩展](#)
- [AWS 扩展 .NET Core 设置](#)
- [AWS 日志记录 .NET](#)
- [Amazon Cognito 身份验证扩展库](#)
- [AWS X-Ray SDK for .NET](#)

其他资源

以下是其它可能有用的资源 :

- [开发人员网](#)
- [AWS Cloud 上的 .NET 开发环境 - 快速入门参考部署](#)
- [Hello, Cloud! 博客](#)
- [AWS白皮书 : 在 AWS 上开发和部署 .NET 应用程序](#)
- [AWS Microservice Extractor for .NET](#)
- [Porting Assistant for .NET](#)
- [AWS 开发工具包和工具参考指南](#)

开始使用 AWS SDK for .NET

要使用AWS SDK for .NET，您需要安装工具链并配置应用程序访问 AWS 服务所需的许多基本内容。其中包括：

- 适当的用户账户或角色
- 该用户账户或代入该角色的用户账户的身份验证信息
- AWS 区域规格
- AWSSDK 程序包或程序集

本部分中的一些主题提供了有关如何配置这些基本内容的信息。

本部分中的其它主题提供了可以配置项目的更高级方式的信息。

主题

- [安装和配置工具链](#)
- [使用 AWS 配置软件开发工具包身份验证](#)
- [快速了解AWS SDK for .NET](#)
- [启动新项目](#)
- [配置 AWS 区域](#)
- [通过 NuGet 安装 AWSSDK 程序包](#)
- [在没有 NuGet 的情况下安装 AWSSDK 程序集](#)
- [凭证和配置文件解析](#)
- [有关用户和角色的其它信息](#)
- [AWS SDK for .NET项目的高级配置](#)
- [使用旧版凭证](#)

安装和配置工具链

要使用AWS SDK for .NET，您必须安装特定的开发工具。

跨平台开发

对于 Windows、Linux 或 macOS 上的跨平台 .NET 开发，需要具备：

- Microsoft [.NET Core 开发工具包](#)，版本 2.1、3.1 或更高版本，其中包括 .NET 命令行界面 (CLI) (`dotnet`) 和 .NET Core 运行时。
- 适合您的操作系统和要求的代码编辑器或集成式开发环境 (IDE)。这通常为 .NET Core 提供一些支持。

示例包括 [Microsoft Visual Studio Code \(VS 代码\)](#)、[JetBrains Rider](#) 和 [Microsoft Visual Studio](#)。

- (可选) 一个适用于您选择的编辑器和操作系统的 AWS 工具包 (如果有)。

示例包括 [AWS Toolkit for Visual Studio Code](#)、[AWS Toolkit for JetBrains](#) 和 [AWS Toolkit for Visual Studio](#)。

在 Windows 上使用 Visual Studio 和 .NET Core 开发

对于在 Windows 使用 Visual Studio 和 .NET Core 进行开发，需要以下项：

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1、3.1 或更高版本

默认情况下，安装最新版本的 Visual Studio 时通常会包含此功能。

- (可选) AWS Toolkit for Visual Studio，这款插件提供了用户界面，可用于从 Visual Studio 管理 AWS 资源和本地配置文件。要安装该工具包，请参阅[AWS Toolkit for Visual Studio 设置](#)。

有关更多信息，请参阅 [AWS Toolkit for Visual Studio 用户指南](#)。

后续步骤

[使用 AWS 配置软件开发工具包身份验证](#)

使用 AWS 配置软件开发工具包身份验证

使用 AWS 服务进行开发时，您必须确定您的代码是如何使用 AWS 进行身份验证的。您可以通过不同的方式对 AWS 资源的编程访问权限进行配置，具体取决于环境和您可用的 AWS 访问权限。

要查看适用于开发工具包的各种身份验证方法，请参阅《AWS 开发工具包和工具参考指南》中的[身份验证和访问](#)。

本主题假设新用户正在本地进行开发工作，雇主未向其提供身份验证方法，新用户将使用 AWS IAM Identity Center 来获取临时凭证。如果您的环境与这些假设不符，则本主题中的某些信息可能不适用于您，或者某些信息可能已经提供给您。

配置此环境需要几个步骤，总结如下：

1. [启用和配置 IAM Identity Center](#)
2. [将开发工具包配置为使用 IAM Identity Center。](#)
3. [开始 AWS 访问门户会话](#)

启用和配置 IAM Identity Center

要使用 IAM Identity Center，必须先启用并进行配置。要详细了解如何为开发工具包执行此操作，请查看《AWS 开发工具包和工具参考指南》中 [IAM Identity Center 身份验证](#) 主题内的步骤 1。具体而言，请按照我没有通过 IAM Identity Center 确立访问权限下的所有必要说明进行操作。

将开发工具包配置为使用 IAM Identity Center。

有关如何将开发工具包配置为使用 IAM Identity Center 的信息，请参阅《AWS 开发工具包和工具参考指南》中 [IAM Identity Center 身份验证](#) 主题内的步骤 2。完成此配置后，您的系统应包含以下元素：

- AWS CLI，您可用于在运行应用程序之前启动 AWS 访问门户会话。
- 该共享 AWS config 文件包含一个 [\[default\] 配置文件](#)，在该配置文件中包含一组配置值，可以从开发工具包中引用这些配置值。要查找此文件的位置，请参阅《AWS 开发工具包和工具参考指南》中的[共享文件的位置](#)。在向 AWS 发送请求之前，AWS SDK for .NET 使用配置文件的 SSO 令牌提供程序获取凭证。sso_role_name 值是与 IAM Identity Center 权限集关联的 IAM 角色，应允许访问您的应用程序中使用的 AWS 服务。

以下示例 config 文件显示了使用 SSO 令牌提供程序设置的默认配置文件。配置文件的 sso_session 设置是指所指定的 sso-session 节。sso-session 节包含启动 AWS 访问门户会话的设置。

```
[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

如果您使用 AWS IAM Identity Center 进行身份验证，您的应用程序必须引用以下 NuGet 程序包，这样 SSO 解析才能生效：

- AWSSDK.SSO
- AWSSDK.SSO0IDC

未能引用这些程序包将导致运行时系统异常。

开始 AWS 访问门户会话

在运行访问 AWS 服务的应用程序之前，需要有活动的 AWS 访问门户会话，以便开发工具包使用 IAM Identity Center 身份验证来解析凭证。根据配置的会话时长，访问权限最终将过期，并且开发工具包将遇到身份验证错误。要登录 AWS 访问门户，请在 AWS CLI 中运行以下命令。

```
aws sso login
```

由于您有默认的配置文件设置，因此无需使用 `--profile` 选项调用该命令。如果您的 SSO 令牌提供程序配置在使用指定的配置文件，则命令为 `aws sso login --profile named-profile`。

要测试您是否已有活动会话，请运行以下 AWS CLI 命令。

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

对此命令的响应应该报告共享 config 文件中配置的 IAM Identity Center 账户和权限集。

Note

如果您已经有一个有效的 AWS 访问门户会话并且运行了 `aws sso login`，则无需提供凭证。

登录过程可能会提示您允许 AWS CLI 访问您的数据。由于 AWS CLI 基于适用于 Python 的 SDK 而构建，因此权限消息可能包含 `botocore` 名称的变体。

其他信息

- 有关在开发环境中使用 IAM Identity Center 和 SSO 的更多信息，请参阅[高级身份验证](#)部分中的[单点登录](#)。此信息包括替代方法和更高级的方法，以及向您展示如何使用这些方法的教程。
- 有关适用于软件开发工具包身份验证的更多选项，例如配置文件和环境变量的使用，请参阅《AWS 开发工具包和工具参考指南》中的[配置](#)章节。
- 有关最佳实践的更多信息，请参阅《IAM 用户指南》中的[IAM 中的安全最佳实践](#)。
- 要创建短期 AWS 凭证，请参阅《IAM 用户指南》中的[临时安全凭证](#)。
- 要了解其他凭证提供程序，请参阅《AWS 开发工具包和工具参考指南》中的[标准化凭证提供程序](#)。

快速了解AWS SDK for .NET

本部分为不熟悉AWS SDK for .NET的开发人员提供了基本教程。

Note

在使用这些教程之前，必须先[安装工具链](#)并[配置开发工具包身份验证](#)。

有关为特定 AWS 服务开发软件及相关代码示例的信息，请参阅[使用 AWS 服务](#)。有关其他代码示例，请参阅[AWS SDK for .NET 代码示例](#)。

主题

- [使用AWS SDK for .NET的简单跨平台应用程序](#)
- [使用AWS SDK for .NET的基于 Windows 的简单应用程序](#)
- [后续步骤](#)

使用AWS SDK for .NET的简单跨平台应用程序

本教程将AWS SDK for .NET与 .NET Core 一起用于跨平台开发。本教程向您展示如何使用开发工具包列出您拥有的 [Amazon S3 桶](#)，并且可以选择创建新桶。

您将使用跨平台工具（例如 .NET 命令行界面 (CLI)）完成本教程。有关配置开发环境的其它方法，请参阅[安装和配置工具链](#)。

对于 Windows、Linux 或 macOS 上的跨平台 .NET 开发，需要：

- Microsoft [.NET Core 开发工具包](#)，版本 2.1、3.1 或更高版本，其中包括 .NET 命令行界面 (CLI) (`dotnet`) 和 .NET Core 运行时。
- 适合您的操作系统和要求的代码编辑器或集成式开发环境 (IDE)。这通常为 .NET Core 提供一些支持。

示例包括 [Microsoft Visual Studio Code \(VS 代码\)](#)、[JetBrains Rider](#) 和 [Microsoft Visual Studio](#)。

Note

在使用这些教程之前，必须先[安装工具链](#)并[配置开发工具包身份验证](#)。

步骤

- [创建项目](#)
- [创建代码](#)
- [运行应用程序](#)
- [清理](#)

创建项目

1. 打开命令提示符或终端。查找或创建可以在其中创建 .NET 项目的操作系统文件夹。
2. 在该文件夹中，运行以下命令以创建 .NET 项目。

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

3. 转到新创建的 S3CreateAndList 文件夹并运行以下命令。

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

前面的命令从 [NuGet Package Manager](#) 中安装 NuGet 程序包。由于我们确切地知道在本教程中所需的 NuGet 程序包，因此现在可以执行这一步。在开发过程中，知道所需的程序包也很常见。出现这种情况时，可以运行类似的命令。

创建代码

1. 在 S3CreateAndList 文件夹中，查找并在代码编辑器中打开 Program.cs。
2. 用以下代码替换内容并保存文件。

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

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOIDC
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\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;
    }
}
}
```

运行应用程序

1. 运行以下命令。

```
dotnet run
```

2. 检查输出以查看您拥有的 Amazon S3 桶数 (如果有) 及其名称。
3. 为新 Amazon S3 桶选择名称。以“dotnet-quicktour-s3-1-cross-”为基础，并添加独特的内容，如 GUID 或您的名字等。请务必遵守存储桶命名规则，如 [Amazon S3 用户指南](#) 中的 [存储桶命名规则](#) 中所述。
4. 运行以下命令，将 **BUCKET-NAME** 替换为您选择的存储桶的名称。

```
dotnet run BUCKET-NAME
```

5. 检查输出以查看创建的新存储桶。

清理

执行本教程时，您创建了一些可选择在此时清理的资源。

- 如果您不想保留应用程序在之前步骤中创建的桶，可以使用位于 <https://console.aws.amazon.com/s3/> 的 Amazon S3 桶将其删除。
- 如果您不想保留您的 .NET 项目，请从开发环境中删除 S3CreateAndList 文件夹。

后续工作

返回[快速指南菜单](#)或直接转到[此快速指南的末尾](#)。

使用AWS SDK for .NET的基于 Windows 的简单应用程序

本教程在 Windows 上使用AWS SDK for .NET以及 Visual Studio 和 .NET Core。本教程向您展示如何使用开发工具包列出您拥有的 [Amazon S3 桶](#)，并且可以选择创建新桶。

您将在 Windows 上使用 Visual Studio 和 .NET Core 完成本教程。有关配置开发环境的其它方法，请参阅[安装和配置工具链](#)。

对于在 Windows 使用 Visual Studio 和 .NET Core 进行开发，需要：

- [Microsoft Visual Studio](#)
- Microsoft .NET Core 2.1、3.1 或更高版本

默认情况下，安装最新版本的 Visual Studio 时通常会包含此功能。

Note

在使用这些教程之前，必须先[安装工具链](#)并[配置开发工具包身份验证](#)。

步骤

- [创建项目](#)
- [创建代码](#)
- [运行应用程序](#)
- [清理](#)

创建项目

1. 打开 Visual Studio 并创建一个使用 C# 版本的控制台应用程序模板的新项目；也就是说，描述为：“... 用于创建可以在 .NET 上运行的命令行应用程序...”。将项目命名为 S3CreateAndList。

Note

不要选择控制台应用程序模板的 .NET Framework 版本，或者，如果选择了，请务必使用 .NET Framework 4.6.2 或更高版本。

2. 加载新创建的项目后，选择“工具”、“Pack NuGet Package Manager”、“管理解决方案 NuGet 包”。
3. 浏览以下 NuGet 软件包并将其安装到项目中：AWSSDK.S3、AWSSDK.SecurityToken、AWSSDK.SSO、和 AWSSDK.SSO0IDC

此过程通过 NuGet 软件包[管理器安装 NuGet 软件包](#)。因为我们确切地知道本教程需要什么 NuGet 软件包，所以我们可以现在执行这个步骤了。在开发过程中，知道所需的程序包也很常见。出现这种情况时，请按照类似的过程安装它们。

4. 如果您打算从命令提示符处运行应用程序，请立即打开命令提示符并导航到将包含构建输出的文件夹。这通常是类似于 S3CreateAndList\S3CreateAndList\bin\Debug\net6.0 的内容，但具体取决于您的环境。

创建代码

1. 在 S3CreateAndList 项目中，查找并在 IDE 中打开 Program.cs。
2. 用以下代码替换内容并保存文件。

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

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

namespace S3CreateAndList
{
```

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

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

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

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

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

```

```
        {
            Console.WriteLine("Caught exception when creating a bucket:");
            Console.WriteLine(e.Message);
        }
    }

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

//
// Method to parse the command line.
private static Boolean GetBucketName(string[] args, out String bucketName)
{
    Boolean retval = false;
    bucketName = String.Empty;
    if (args.Length == 0)
    {
        Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
            "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
        bucketName = String.Empty;
        retval = false;
    }
    else if (args.Length == 1)
    {
        bucketName = args[0];
        retval = true;
    }
    else
    {
        Console.WriteLine("\nToo many arguments specified." +
            "\n\n dotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
            "\n\n Usage: 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. 构建应用程序。

Note

如果您使用的是旧版 Visual Studio，则可能会遇到类似以下内容的构建错误：
“Feature 'async main' is not available in C# 7.0. Please use language version 7.1 or greater.”

如果您遇到此错误，请将项目设置为使用该语言的更高版本。这通常在项目属性构建、高级中完成。

运行应用程序

1. 不带命令行参数运行应用程序。在命令提示符中（如果之前已打开）或从 IDE 执行此操作。
2. 检查输出以查看您拥有的 Amazon S3 桶数（如果有）及其名称。
3. 为新 Amazon S3 桶选择名称。使用“dotnet-quicktour-s3-1-winvs-”作为基础，然后为其添加一些独特的东西，例如 GUID 或你的名字。请务必遵守存储桶命名规则，如 [Amazon S3 用户指南](#) 中的 [存储桶命名规则](#) 中所述。
4. 再次运行应用程序，这次提供存储桶名称。

在命令行中，将以下命令中的 **BUCKET-NAME** 替换为您选择的桶的名称：

```
S3CreateAndList BUCKET-NAME
```

或者，如果您在 IDE 中运行应用程序，请选择“项目”、“S3 CreateAndList 属性”、“调试”，然后在那里输入存储桶名称。

5. 检查输出以查看创建的新存储桶。

清理

执行本教程时，您创建了一些可选择在此时清理的资源。

- 如果您不想保留应用程序在之前步骤中创建的桶，可以使用位于 <https://console.aws.amazon.com/s3/> 的 Amazon S3 桶将其删除。
- 如果您不想保留您的 .NET 项目，请从开发环境中删除 S3CreateAndList 文件夹。

后续工作

返回 [快速指南菜单](#) 或直接转到 [此快速指南的末尾](#)。

后续步骤

请务必清理您在完成这些教程时创建的所有遗留资源。这包括 AWS 资源或开发环境中的资源，例如文件和文件夹。

既然已经全面了解了 AWS SDK for .NET，您可能想 [启动项目](#)。

启动新项目

您可以使用多种技术来启动访问 AWS 服务的新项目。以下是一些此类技术：

- 如果您不熟悉基于 AWS 的 .NET 开发，或者至少不熟悉 AWS SDK for .NET，则可以在[快速了解](#)中查看完整的示例。其中提供了对 SDK 的简单介绍。
- 您可以使用 .NET CLI 启动基本项目。要查看此示例，请打开命令提示符或终端，创建文件夹或目录并导航到该文件夹或目录，然后输入以下内容。

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

已创建一个空项目，您可以向其中添加代码和 NuGet 程序包。有关更多信息，请参阅[.NET Core 指南](#)。

要查看项目模板列表，请使用以下命令：`dotnet new --list`

- AWS Toolkit for Visual Studio 包括适用于各种 AWS 服务的 C# 项目模板。当您在 Visual Studio 中[安装工具包](#)后，您可以在创建新项目时访问这些模板。

要查看此内容，请转到[AWS Toolkit for Visual Studio 用户指南](#)中的[使用 AWS 服务](#)。该部分中的几个示例创建新项目。

- 如果您在 Windows 上使用 Visual Studio 进行开发，但没有使用 AWS Toolkit for Visual Studio，请使用典型的方法来创建新项目。

要查看示例，请打开 Visual Studio，然后选择文件、新建、项目。搜索“.net core”，然后选择 C# 版本的控制台应用程序 (.NET Core) 或 WPF 应用程序 (.NET Core) 模板。已创建一个空项目，您可以向其中添加代码和 NuGet 程序包。

您可以在[带有指导的代码示例](#)中找到一些有关如何使用 AWS 服务的示例。

Important

如果您使用 AWS IAM Identity Center 进行身份验证，您的应用程序必须引用以下 NuGet 程序包，这样 SSO 解析才能生效：

- AWSSDK.SSO
- AWSSDK.SS00IDC

未能引用这些程序包将导致运行时系统异常。

配置 AWS 区域

使用 AWS 区域可以访问实际位于特定地理区域的 AWS 服务。它可用于保证冗余，并保证您的数据和应用程序接近您和用户访问它们的位置。

要查看当前各 AWS 服务所有支持的区域和端点列表，请参阅 [AWS 一般参考](#) 中的区域和端点。要查看现有区域端点的列表，请参阅 [AWS 服务终端节点](#)。要查看有关区域的详细信息，请参阅 [指定您的账户可以使用哪些 AWS 区域](#)。

您可以创建转到 [特定区域](#) 的 AWS 服务客户端。您还可以为应用程序配置一个将用于 [所有 AWS 服务客户端](#) 的区域。接下来将解释这两种情况。

创建具有特定区域的服务客户端

您可以为应用程序中的任何 AWS 服务客户端指定区域。以此方式设置区域优先于该特定服务客户端的任何全局设置。

现有区域

此示例向您展示了如何在现有区域中实例化 [Amazon EC2 客户端](#)。它使用已定义的 [RegionEndpoint](#) 字段。

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

使用 RegionEndpoint 类的新区域

此示例向您展示了如何使用 [RegionEndpoint.GetBySystemName](#) 构造新的区域端点。

```
var newRegion = RegionEndpoint.GetBySystemName("us-west-new");
```

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

使用服务客户端配置类的新区域

此示例向您展示如何使用服务客户端配置类的 `ServiceURL` 属性来指定区域；在本例中是使用 [AmazonEC2Config](#) 类。

即使区域端点未遵循正规区域端点模式，此技术仍适用。

```
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
}
```

为所有服务客户端指定区域

您可以通过多种方式为应用程序创建的所有 AWS 服务客户端指定区域。此区域用于不是通过特定区域创建的服务客户端。

AWS SDK for .NET按以下顺序查找区域值。

配置文件

在您的应用程序或软件开发工具包已加载的配置文件中进行设置。有关更多信息，请参阅[凭证和配置文件解析](#)。

环境变量

在 `AWS_REGION` 环境变量中设置。

在 Linux 或 macOS 上：


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

在 Windows 上：

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

Note

如果您为整个系统设置此环境变量（使用 `export` 或 `setx`），则它会影响所有 SDK 和工具包，而不仅仅是 AWS SDK for .NET。

AWSConfigs 类

设置为 [AWSConfigs.AWSRegion](#) 属性。

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

区域解析

如果没有使用上述方法来指定 AWS 区域，则 AWS SDK for .NET 会尝试查找 AWS 服务客户端要运行的区域。

区域解析顺序

1. 应用程序配置文件，例如 `app.config` 和 `web.config`。
2. 环境变量（`AWS_REGION` 和 `AWS_DEFAULT_REGION`）。
3. 名称由 `AWSConfigs.AWSProfileName` 中的值指定的配置文件。
4. 名称由 `AWS_PROFILE` 环境变量指定的配置文件。
5. `[default]` 配置文件。
6. Amazon EC2 实例元数据（如果在 EC2 实例上运行）。

如果未找到区域，软件开发工具包会引发异常，指出 AWS 服务客户端没有配置区域。

有关中国（北京）区域的特殊信息

要使用中国（北京）区域中的服务，您必须拥有特定于中国（北京）区域的账户和凭证。其他 AWS 区域的账户和凭证不适用于中国（北京）区域。同样，中国（北京）区域的账户和凭证也不适用于其他 AWS 区域。有关可在中国（北京）区域使用的端点和协议的信息，请参阅[中国（北京）区域端点](#)。

有关新 AWS 服务的特殊信息

新的 AWS 服务会首先在少数几个区域中推出，然后在其它区域中支持。在这些情况下，您无需安装最新的软件开发工具包来访问该服务的新区域。您可以按各个客户端或者按全球来指定新添加的区域，如之前所示。

通过 NuGet 安装 AWSSDK 程序包

[NuGet 是 .NET 平台的包管理系统](#)。有了 NuGet，您可以将 [AWSSDK 程序包](#) 以及其它几个扩展安装到您的项目中。有关其它信息，请参阅 GitHub 网站上的 [aws/dotnet](#) 存储库。

NuGet 始终具备最新版本的 AWSSDK 程序包，以及之前的版本。NuGet 能够识别程序包之间的依赖关系并自动安装所有必需的程序包。

Warning

NuGet 程序包列表可能包括一个名为“AWSSDK”的程序包（没有附加标识符）。请勿安装此 NuGet 程序包；它是旧版，不应用于新项目。

通过 NuGet 安装的程序包与您的项目一起存储，而不是存储在中心位置。这使您可以安装特定于指定应用程序的程序集版本，而不会造成其他应用程序的兼容性问题。有关 NuGet 的更多信息，请参阅[NuGet 文档](#)。

Note

如果您不能或不允许按项目下载和安装 NuGet 程序包，则可以获取 AWSSDK 程序集并将其存储在本地。

如果这适用于您，并且您尚未获取 AWSSDK 程序集，请参阅[获取 AWSSDK 程序集](#)。要了解如何使用本地存储的程序集，请参阅[在没有 NuGet 的情况下安装 AWSSDK 程序集](#)。

从命令行提示符处或终端使用 NuGet

1. 转到 [AWSSDK NuGet 程序包](#)，并确定您在项目中需要哪些 NuGet 程序包；例如 [AWSSDK.S3](#)。
2. 从该程序包的网页复制 .NET CLI 命令，如以下示例所示。

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

3. 在项目的目录中运行该 .NET CLI 命令。NuGet 还会安装所有依赖项，例如 [AWSSDK.Core](#)。

Note

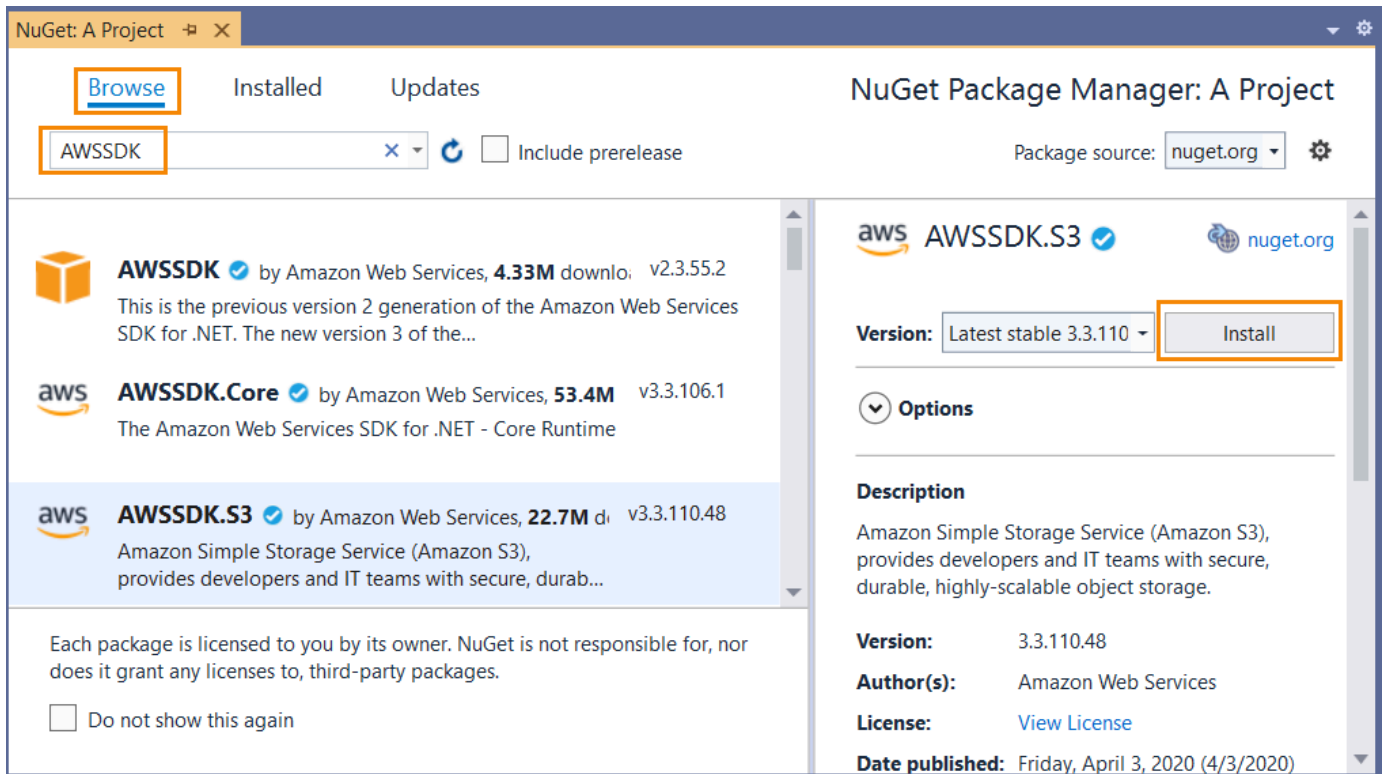
如果您只需要最新版本的 NuGet 程序包，则可以从命令中排除版本信息，如以下示例所示。

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

从 Visual Studio Solution Explorer 中使用 NuGet

1. 在 Solution Explorer 中，右键单击您的项目，然后从上下文菜单中选择 Manage NuGet Packages。
2. 在 NuGet Package Manager (NuGet 软件包管理器) 的左窗格中，选择 Browse (浏览)。然后，您可以使用搜索框搜索所要安装的软件包。NuGet 还会安装所有依赖项，例如 [AWSSDK.Core](#)。

下图显示了 AWSSDK.S3 程序包的安装。



从 Package Manager Console 使用 NuGet

从 Visual Studio 工具菜单，依次选择 NuGet Package Manager 和 Package Manager Console。

您可以从 Package Manager Console，使用 **Install-Package** 命令安装所需的 AWSSDK 程序包。例如，要安装 [AWSSDK.S3](#)，请使用以下命令。

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

NuGet 还会安装所有依赖项，例如 [AWSSDK.Core](#)。

如果您需要安装程序包的早期版本，请使用 `-Version` 选项并指定所需的程序包版本，如下示例所示。

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

有关 Package Manager Console 命令的更多信息，请参阅微软 [NuGet 文档](#) 中的 [PowerShell 参考](#)。

在没有 NuGet 的情况下安装 AWSSDK 程序集

本主题介绍如何使用您在本地获取并存储的 AWSSDK 程序集，如[获取 AWSSDK 程序集](#)中所述。这不是处理软件开发工具包参考的推荐方法，但在某些环境中却是必需的。

Note

处理软件开发工具包参考的推荐方法是仅下载并安装每个项目所需的 NuGet 程序包。[通过 NuGet 安装 AWSSDK 程序包](#)中介绍了这种方法。

安装 AWSSDK 程序集

1. 在项目区域中为所需的 AWSSDK 程序集创建一个文件夹。举个例子，您可以调用这个文件夹 `AwsAssemblies`。
2. 如果您尚未这样做，请[获取 AWSSDK 程序集](#)，它将程序集放在某个本地下载或安装文件夹中。将所需程序集的 DLL 文件从该下载文件夹复制到您的项目中（在我们的示例中为 `AwsAssemblies` 文件夹）。

还请务必复制所有依赖关系。您可以在 [GitHub](#) 网站上找到有关依赖关系的信息。

3. 请按如下方式引用所需的程序集。

Cross-platform development

1. 打开项目的 `.csproj` 文件并添加一个 `<ItemGroup>` 元素。
2. 在 `<ItemGroup>` 元素中，为每个必需的程序集添加一个具有 `Include` 属性的 `<Reference>` 元素。

例如，对于 Amazon S3，您可以在项目 `.csproj` 文件中添加以下几行。

在 Linux 和 macOS 上：

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

在 Windows 上：

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

3. 保存项目的 .csproj 文件。

Windows with Visual Studio and .NET Core

1. 在 Visual Studio 中，加载您的项目并打开项目、添加引用。
2. 选择对话框底部的浏览按钮。导航到项目的文件夹并将所需的 DLL 文件复制到子文件夹（例如 AwsAssemblies）。
3. 选择所有 DLL 文件，选择添加，然后选择确定。
4. 保存您的项目。

凭证和配置文件解析

AWS SDK for .NET按特定顺序搜索凭证，并对当前应用程序使用第一个可用集。

凭证搜索顺序

1. 在 AWS 服务客户端上显式设置的凭证，如[访问应用程序中的凭证和配置文件](#)中所述。

Note

该主题之所以出现在[特殊注意事项](#)部分中，是因为它不是指定凭证的首选方法。

2. 一个凭证配置文件，其名称由 [AWSConfigs.AWSProfileName](#) 中的值指定。
3. 名称由 AWS_PROFILE 环境变量指定的凭证配置文件。
4. [default] 凭证配置文件。
5. 根据 AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY 和 AWS_SESSION_TOKEN 环境变量 (如果这些变量都不为空) 创建的 [SessionAWSCredentials](#)。
6. 根据 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量 (如果这两个变量不为空) 创建的 [BasicAWSCredentials](#)。
7. Amazon ECS 任务的[任务的 IAM 角色](#)。
8. Amazon EC2 实例元数据。

如果您的应用程序在 Amazon EC2 实例上运行（例如在生产环境中），请使用 IAM 角色，如 [使用 IAM 角色授予访问权限](#) 中所述。否则，例如在预发布测试中，将凭证存储在服务器上使用 AWS 凭证文件格式的文件中（Web 应用程序有权访问此文件）。

配置文件解析

凭证有两种不同的存储机制，请务必了解如何配置 AWS SDK for .NET 以使用它们。[AWSConfigs.AWSProfilesLocation](#) 属性控制 AWS SDK for .NET 如何查找凭证配置文件。

AWSProfilesLocation	配置文件解析行为
null (未设置) 或空	如果平台支持，请搜索 SDK Store，然后在 默认位置 搜索共享 AWS 凭证文件。如果配置文件不在上述任何一个位置，请搜索 <code>~/.aws/config</code> （Linux 或 macOS）或 <code>%USERPROFILE%\aws\config</code> （Windows）。
AWS 凭证文件格式的文件的完整路径	仅在指定文件中搜索具有指定名称的配置文件。

使用联合用户账户凭证

对于采用 AWS SDK for .NET（[AWSSDK.Core](#) 版本 3.1.6.0 和更高版本）的应用程序，可以通过 Active Directory Federation Services（AD FS）使用联合用户账户来利用安全断言标记语言（SAML）访问 AWS 服务。

联合访问支持意味着用户可使用您的 Active Directory 进行身份验证。系统将自动为用户授予临时凭证。当您的应用程序调用 AWS 服务时，将使用这些临时凭证（有效期为 1 小时）。开发工具包将管理临时凭证。对于已加入域的用户账户，如果您的应用程序发出调用但凭证已过期，则将自动重新验证用户的身份并授予全新凭证。（对于未加入域的账户，在重新进行身份验证之前，系统将提示用户输入凭证。）

要在 .NET 应用程序中使用此支持，您必须先使用 PowerShell cmdlet 设置角色配置文件。要了解具体信息，请参阅[AWS Tools for Windows PowerShell 文档](#)。

设置角色配置文件后，请在应用程序中引用该配置文件。有多种方法可以做到这一点，其中一种是像处理其他凭证配置文件一样使用 [AWSConfigs.AWSProfileName](#) 属性。

AWS Security Token Service 程序集 ([awssdk.securityToken](#)) 提供 SAML 支持以获取 AWS 凭证。要使用联合用户账户凭证，请确保您的应用程序可以使用此程序集。

指定角色或临时凭证

对于 Amazon EC2 实例上运行的应用程序，管理凭证最安全的方式是使用 IAM 角色，如[使用 IAM 角色授予访问权限](#)中所述。

对于软件可执行文件对组织外部用户可用的应用程序情景，建议您将软件设计为使用临时安全凭证。除了提供对 AWS 资源的有限访问之外，让这些凭证在指定时段后过期也会有好处。有关临时安全凭证的更多信息，请参阅：

- [临时安全凭证](#)
- [Amazon Cognito 身份池](#)

使用代理凭证

如果您的软件通过代理与 AWS 进行通信，则可使用服务的 Config 类的 ProxyCredentials 属性来为代理指定凭证。服务的 Config 类通常是该服务主命名空间的一部分。示例包括以下内容：[Amazon.CloudDirectory](#) 命名空间中的 [AmazonCloudDirectoryConfig](#) 和 [Amazon.GameLift](#) 命名空间中的 [AmazonGameLiftConfig](#)。

例如，对于 [Amazon S3](#)，您可以使用如下所示的代码，其中 SecurelyStoredUserName 和 SecurelyStoredPassword 为 [NetworkCredential](#) 对象中指定的代理用户名和密码。

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

Note

早期版本的开发工具包使用的是 ProxyUsername 和 ProxyPassword，但这两个属性已被弃用。

有关用户和角色的其它信息

要在 AWS 上进行 .NET 开发或在 AWS 上运行 .NET 应用程序，您需要具备适合这些任务的用户、权限集和服务角色的组合。

您创建的特定用户、权限集和服务角色以及使用它们的方式，将取决于您应用程序的要求。下面的一些附加信息介绍了可能使用它们的原因以及如何创建它们。

用户和权限集

尽管可以使用具有长期凭证的 IAM 用户账户来访问 AWS 服务，但这已不再是最佳实践，应予以避免。即使在开发过程中，最佳做法也是在 AWS IAM Identity Center 中创建用户和权限集，并使用身份源提供的临时凭证。

对于开发，您可以使用自己创建的或在[配置开发工具包身份验证](#)中提供的用户。如果您拥有适当的 AWS Management Console 权限，还可以为该用户创建具有最低权限的不同权限集，或者创建专用于开发项目的新用户，同时提供具有最低权限的权限集。您选择的行动方案（如果有）取决于您的情况。

有关这些用户和权限集以及如何创建它们的更多信息，请参阅《AWS 开发工具包和工具参考指南》中的[身份验证和访问](#)以及《AWS IAM Identity Center 用户指南》中的[入门](#)。

服务角色

您可以设置 AWS 服务角色来代表用户访问 AWS 服务。如果有多人远程运行您的应用程序，则这种访问方式是合适的；例如，在您为此目的创建的 Amazon EC2 实例上。

创建服务角色的过程因情况而异，但基本上如下所示。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 选择 角色，然后选择 创建角色。
3. 选择 AWS 服务，找到并选择 EC2（例如），然后选择 EC2 使用案例（例如）。
4. 选择下一步: 权限，然后为您的应用程序将使用的 AWS 服务选择[适当的策略](#)。

Warning

请勿选择 AdministratorAccess 策略，因为该策略允许对您账户中的几乎所有内容具有读写权限。

5. 选择下一步: 标签，然后输入所需的任何标签。

您可以在 [IAM 用户指南](#) 中的 [使用 AWS 资源标签控制访问](#) 中找到标签相关的信息。

6. 选择下一步: 查看，并提供角色名称和角色描述。然后选择创建角色。

在 [IAM 用户指南](#) 中的 [身份 \(用户、用户组和角色\)](#) 中，您可以找到有关 IAM 角色的高级信息。在 [IAM 角色](#) 主题中查找有关角色的详细信息。

有关角色的其它信息

- 对于 Amazon Elastic Container Service (Amazon ECS)，使用 [适用于任务的 IAM 角色](#)。
- 对于在 Amazon EC2 实例上运行的应用程序，使用 [IAM 角色](#)。

AWS SDK for .NET项目的高级配置

本部分中的主题包含有关您可能感兴趣的其它配置任务和方法的信息。

主题

- [使用 AWSSDK.Extensions.NETCore.Setup 和 IConfiguration 界面](#)
- [配置其他应用程序参数](#)
- [AWS SDK for .NET的配置文件参考](#)

使用 AWSSDK.Extensions.NETCore.Setup 和 IConfiguration 界面

(本主题之前的标题是“使用 .NET Core 配置 AWS SDK for .NET”)

.NET Core 中最大的改变之一是删除了以前在 .NET Framework 和 ASP.NET 应用程序中使用的 ConfigurationManager 以及标准 app.config 和 web.config 文件。

.NET Core 中的配置基于配置提供商建立的键/值对。配置提供商将配置数据从各种配置源读取为键/值对，包括命令行参数、目录文件、环境变量和设置文件。

Note

有关更多信息，请参阅 [ASP.NET Core 中的配置](#)。

为了便于将 AWS SDK for .NET 与 .NET Core 结合使用，您可以使用 [AWSSDK.Extensions.NETCore.Setup](#) NuGet 程序包。与许多 .NET Core 库一样，它将扩展方法添加到 AWS 接口，以实现无缝获取 IConfiguration 配置。

使用 AWSSDK.Extensions.NETCore.Setup

假设您创建了一个 ASP.NET Core 模型视图控制器 (MVC) 应用程序，该应用程序可以使用 Visual Studio 中的 ASP.NET Core Web Application 模板或在 .NET Core CLI 中运行 `dotnet new mvc ...` 来完成。创建此类应用程序时，`Startup.cs` 的构造函数通过从配置提供商读取各种输入源来处理配置，例如读取 `appsettings.json`。

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

要使用 `Configuration` 对象获取 AWS 选项，请先添加 `AWSSDK.Extensions.NETCore.Setup` NuGet 程序包。然后，将选项添加到下面所述的配置文件中。

请注意，添加到您项目中的其中一个文件是 `appsettings.Development.json`。这就代表 `EnvironmentName` 设置为 `Development`。在开发过程中，将配置放入此文件中，该文件只能在本地测试期间读取。在您部署将 `EnvironmentName` 设置为 `Production` 的 Amazon EC2 实例时，将忽略此文件并且 AWS SDK for .NET 回退到为 Amazon EC2 实例配置的 IAM 凭证和区域。

以下配置设置显示您可以添加到您项目的 `appsettings.Development.json` 文件中用于提供 AWS 设置的值的示例。

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

要访问 CSHTML 文件中的设置，请使用 `Configuration` 指令：

```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

<h1>Contact</h1>

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

```
</p>
```

要访问在文件中从代码设置的 AWS 选项，请调用在 `IConfiguration` 上添加的 `GetAWSSOptions` 扩展方法。

要从这些选项构造服务客户端，请调用 `CreateServiceClient`。以下示例代码说明了如何创建 Amazon S3 服务客户端。（请务必将 [AWSSDK.S3](#) NuGet 程序包添加到您的项目）。

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

您还可以使用 `appsettings.Development.json` 文件中的多个条目创建具有不兼容设置的多个服务客户端，如以下示例所示，其中 `service1` 的配置包括 `us-west-2` 区域，`service2` 的配置包括特殊端点 URL。

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

然后，您可以使用 JSON 文件中的条目获取特定服务的选项。例如，要获取 `service1` 的设置，请使用以下项目。

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

`appsettings` 文件中允许的值

以下应用程序配置值可以在 `appsettings.Development.json` 文件中设置。字段名必须使用所示的大小写。有关这些设置的详细信息，请参阅 [AWS.Runtime.ClientConfig](#) 类。

- 区域
- 配置文件
- ProfilesLocation

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

ASP.NET Core 依赖关系注入

AWSSDK.Extensions.NETCore.Setup NuGet 程序包还集成了 ASP.NET 内核中的新依赖关系注入系统。应用程序的 Startup 类中的 ConfigureServices 方法是添加 MVC 服务的位置。如果应用程序正在使用实体框架，则这也是进行初始化的位置。

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

Note

有关 .NET Core 中依赖关系注入的背景信息在 [.NET Core 文档网站](#) 上提供。

以下 AWSSDK.Extensions.NETCore.Setup NuGet 程序包将新的扩展方法添加到 IServiceCollection，您可用将 AWS 服务添加到依赖关系注入。以下代码演示如何添加从

IConfiguration 读取的 AWS 选项，用于将 Amazon S3 和 DynamoDB 添加到服务列表中。（请务必将 [AWSSDK.S3](#) 和 [AWSSDK.DynamoDBv2](#) 程序包添加到您的项目）。

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

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

现在，如果您的 MVC 控制器使用 IAmazonS3 或 IAmazonDynamoDB 作为其构造函数中的参数，则依赖关系注入系统会传入这些服务。

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

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

    ...
}
```

配置其他应用程序参数

Note

本主题中的信息特定于基于 .NET Framework 的项目。默认情况下，App.config 和 Web.config 文件不存在于基于 .NET Core 的项目中。

打开查看 .NET Framework 内容

可配置多种应用程序参数：

- [AWSLogging](#)
- [AWSLogMetrics](#)
- [AWSRegion](#)
- [AWSResponseLogging](#)
- [AWS.DynamoDBContext.TableNamePrefix](#)
- [AWS.S3.UseSignatureVersion4](#)
- [AWSEndpointDefinition](#)
- [AWS 服务生成的端点](#)

这些参数可以在应用程序的 App.config 或 Web.config 文件中配置。虽然您也可以使用 AWS SDK for .NET API 配置这些参数，但建议您使用应用程序的 .config 文件。此处介绍了这两种方法。

有关本主题后文中所述的使用 <aws> 元素的更多信息，请参阅[AWS SDK for .NET 配置文件参考](#)。

AWSLogging

配置开发工具包应如何记录事件 (如果记录)。例如，建议的方法是使用 <logging> 元素，该元素是 <aws> 元素的子元素：

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

或者：

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

可能的值包括：

None

禁用事件日志记录。这是默认模式。

log4net

使用 log4net 记录。

SystemDiagnostics

使用 System.Diagnostics 类记录。

您可以为 `logTo` 属性设置多个值，以逗号分隔。以下示例在 `.config` 文件中设置 `log4net` 和 `System.Diagnostics` 日志记录：

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

或者：

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

或者，使用 AWS SDK for .NET API，将 [LoggingOptions](#) 枚举的值合并，然后设置 [AWSConfigs.Logging](#) 属性：

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

对此设置的更改仅对新 AWS 客户端实例生效。

AWSLogMetrics

指定开发工具包是否应记录性能指标。要在 `.config` 文件中设置指标日志记录配置，请在 `<logging>` 元素中设置 `logMetrics` 属性值，该元素是 `<aws>` 元素的子元素：

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

此外，在 `<appSettings>` 部分中设置 `AWSLogMetrics` 关键字：

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

此外，要使用 AWS SDK for .NET API 设置指标日志记录，请设置 [AWSConfigs.LogMetrics](#) 属性：

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

对于所有客户端/配置，此设置用于配置默认 `LogMetrics` 属性。对此设置的更改仅对新 AWS 客户端实例生效。

AWSRegion

为没有明确指定区域的客户端配置默认 AWS 区域。要在 `.config` 文件中设置区域，建议的方法是在 `aws` 元素中设置 `region` 属性值：


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

此外，在 `部分` 中设置 `AWSRegion<appSettings>` 关键字：

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

或者，要使用 AWS SDK for .NET API 设置区域，请设置 [AWSConfigs.AWSRegion](#) 属性：

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

有关为特定区域创建 AWS 客户端的更多信息，请参阅 [AWS 区域选择](#)。对此设置的更改仅对新 AWS 客户端实例生效。

AWSResponseLogging

配置开发工具包在什么情况下记录服务响应。可能的值包括：

Never

从不记录服务响应。这是默认模式。

Always

始终记录服务响应。

OnError

仅在出错时记录服务响应。

要在 `.config` 文件中设置服务日志记录配置，推荐的方法是在 `<logging>` 元素中设置 `logResponses` 属性值，该元素是 `<aws>` 元素的子元素：

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

此外，在 `部分` 中设置 `AWSResponseLogging<appSettings>` 关键字：

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

或者，要使用AWS SDK for .NET API 设置服务日志记录，请将 [AWSConfigs.ResponseLogging](#) 属性设置为 [ResponseLoggingOption](#) 枚举的值之一：

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

对此设置的更改会立即生效。

AWS.DynamoDBContext.TableNamePrefix

配置 `TableNamePrefix` 将使用的默认 `DynamoDBContext` (在未手动配置的情况下)。

要在 `.config` 文件中设置表名前缀，建议的方法是在 `<dynamoDBContext>` 元素中设置 `tableNamePrefix` 属性值，该元素是 `<dynamoDB>` 元素 (这又是 `<aws>` 的子元素) 的子元素：

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

此外，在 `<appSettings>` 部分中设置 `AWS.DynamoDBContext.TableNamePrefix` 关键字：

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

或者，要使用AWS SDK for .NET API 设置表名前缀，请设置 [AWSConfigs.DynamoDBContextTableNamePrefix](#) 属性：

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

对此设置的更改仅在新构建的 `DynamoDBContextConfig` 和 `DynamoDBContext` 实例中生效。

AWS.S3.UseSignatureVersion4

配置 Amazon S3 客户端是否应使用签名版本 4 对请求签名。

要在 `.config` 文件中为 Amazon S3 设置签名版本 4 的签名，推荐的方法是在 `<s3>` 元素中设置 `useSignatureVersion4` 属性，该元素是 `<aws>` 元素的子元素：

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

此外，在 `<appSettings>` 部分中将 `AWS.S3.UseSignatureVersion4` 密钥设置为 `true`：

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

或者，要使用AWS SDK for .NET API 设置签名版本 4 签名，请将 [AWSConfigs.S3UseSignatureVersion4](#) 属性设置为 true：

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

默认情况下，此设置是 false，但在某些情况下或者在某些区域中，默认会使用签名版本 4。当设置为 true 时，签名版本 4 将用于所有请求。对此设置的更改仅对新 Amazon S3 客户端实例生效。

AWSEndpointDefinition

配置开发工具包是否应使用自定义配置文件，该文件定义区域和终端节点。

要在 .config 文件中设置终端节点定义文件，建议在 <aws> 元素中设置 endpointDefinition 属性值。

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

此外，您可以在 部分中设置 `AWSEndpointDefinition<appSettings>` 关键字：

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

或者，要使用AWS SDK for .NET API 设置终端节点定义文件，请设置 [AWSConfigs.EndpointDefinition](#) 属性：

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

如果没有提供文件名，则不使用自定义配置文件。对此设置的更改仅对新 AWS 客户端实例生效。endpoint.json 文件可从<https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json>中获得。

AWS 服务生成的端点

一些 AWS 服务生成自己的端点，而不使用区域端点。这些服务的客户端使用特定于该服务的服务 URL 以及您的资源。这些服务的两个示例是 Amazon CloudSearch 和 AWS IoT。以下示例演示如何获取这些服务的终端节点。

Amazon CloudSearch 端点示例

Amazon CloudSearch 客户端用于访问 Amazon CloudSearch 配置服务。您使用 Amazon CloudSearch 配置服务来创建、配置和管理搜索域。要创建搜索域，请创建 [CreateDomainRequest](#)

对象并提供 `DomainName` 属性。使用请求对象创建 [AmazonCloudSearchClient](#) 对象。调用 [CreateDomain](#) 方法。调用返回的 [CreateDomainResponse](#) 对象包含 `DomainStatus` 属性，该属性具有 `DocService` 和 `SearchService` 终端节点。创建 [AmazonCloudSearchDomainConfig](#) 对象并使用它来初始化 `DocServiceAmazonCloudSearchDomainClientSearchService` 类的 [和](#) 实例。

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

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

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

// Test the SearchService endpoint
var searchServiceConfig = new AmazonCloudSearchDomainConfig
{
```

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

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

AWS IoT 终端节点示例

要获取 AWS IoT 的终端节点，请创建 [AmazonIoTClient](#) 对象并调用 [DescribeEndPoint](#) 方法。返回的 [DescribeEndPointResponse](#) 对象包含 `EndpointAddress`。创建 [AmazonIotDataConfig](#) 对象，设置 `ServiceURL` 属性，并使用对象实例化 [AmazonIotDataClient](#) 类。

```
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");
}
}
```

AWS SDK for .NET的配置文件参考

Note

本主题中的信息特定于基于 .NET Framework 的项目。默认情况下，App.config 和 Web.config 文件不存在于基于 .NET Core 的项目中。

打开查看 .NET Framework 内容

您可以使用 .NET 项目的 App.config 或 Web.config 文件指定 AWS 设置（如 AWS 凭证、日志记录选项、AWS 服务端点和 AWS 区域）以及某些 AWS 服务设置（例如 Amazon DynamoDB、Amazon EC2 和 Amazon S3）。以下信息介绍了如何正确设置 App.config 或 Web.config 文件的格式，以指定这些设置类型。

Note

虽然您可以继续在 App.config 或 Web.config 文件中使用 <appSettings> 元素来指定 AWS 设置，但我们建议您使用 <configSections> 和 <aws> 元素，如本主题后面所述。有关 <appSettings> 元素的更多信息，请参阅[配置AWS SDK for .NET应用程序](#)中的元素示例。

Note

虽然您可以继续使用代码文件中的以下 [AWSConfigs](#) 类属性来指定 AWS 设置，但以下属性已被弃用，并且在将来版本中不受支持：

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

通常，我们建议不要使用代码文件中的 [AWSConfigs](#) 类属性来指定 AWS 设置，而应使用 <aws> 或 Web.config 文件中的 <configSections> 和 App.config 元素来指定 AWS

设置，如本主题后面所述。有关上述属性的更多信息，请参阅[配置AWS SDK for .NET应用程序](#)中的 AWSConfigs 代码示例。

主题

- [声明 AWS 设置部分](#)
- [允许的元素](#)
- [元素参考](#)

声明 AWS 设置部分

您可以通过 `<aws>` 元素指定 App.config 或 Web.config 文件中的 AWS 设置。在开始使用 `<aws>` 元素之前，您必须先创建一个 `<section>` 元素 (`<configSections>` 元素的子元素) 并将其 `name` 属性和 `aws` 属性分别设置为 `type` 和 `Amazon.AWSSection, AWSSDK.Core` (如以下示例所示)：

```
<?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>
```

Visual Studio 编辑器提供针对 `<aws>` 元素或其子元素的自动代码完成 (IntelliSense)。

调用 `<aws>` 方法可帮助您创建 `Amazon.AWSConfigs.GenerateConfigTemplate` 元素格式正确的版本。这将以美观打印字符串形式输出 `<aws>` 元素的规范版本，可调整该字符串以满足您的需求。以下部分介绍 `<aws>` 元素的属性和子元素。

允许的元素

以下是 AWS 设置部分中允许的元素之间的逻辑关系列表。要生成此列表的最新版本，您可以调用 `Amazon.AWSConfigs.GenerateConfigTemplate` 方法，这将以字符串形式输出 `<aws>` 元素的规范版本，可调整该字符串以满足您的需求。

```
<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>
```


元素参考

下面是 AWS 设置部分允许的元素列表。对于每个元素，将列出其允许的属性和父-子元素。

主题

- [别名](#)
- [AWS](#)
- [dynamoDB](#)
- [dynamoDBContext](#)
- [ec2](#)
- [日志记录](#)
- [映射](#)
- [property](#)
- [proxy](#)
- [S3](#)

别名

`<alias>` 元素表示集合中的一个项，该集合包含一个或多个源表到目标表的映射，用于指定与为某一类型配置的表不同的表。此元素将从AWS SDK for .NET中的 `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` 属性映射到 `Amazon.Util.TableAlias` 类的实例。在应用表名称前缀之前，将执行重新映射。

此元素可以包含以下属性：

fromTable

源表到目标表的映射的源表部分。此属性映射到AWS SDK for .NET中的 `Amazon.Util.TableAlias.FromTable` 属性。

toTable

源表到目标表的映射的目标表部分。此属性映射到AWS SDK for .NET中的 `Amazon.Util.TableAlias.ToTable` 属性。

`<alias>` 元素的父级为 `<tableAliases>` 元素。

`<alias>` 元素不包含子元素。

以下是使用中的 `<alias>` 元素的示例：

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

AWS

`<aws>` 元素表示 AWS 设置部分中的最顶层元素。此元素可以包含以下属性：

endpointDefinition

自定义配置文件的绝对路径，该文件定义要使用的 AWS 区域和端点。此属性映射到AWS SDK for .NET中的 `Amazon.AWSConfigs.EndpointDefinition` 属性。

profileName

存储的 AWS 凭证的配置文件名称，该凭证将用于进行服务调用。此属性映射到AWS SDK for .NET 中的 `Amazon.AWSConfigs.AWSProfileName` 属性。

profilesLocation

与其它 AWS 开发工具包共享的凭证文件位置的绝对路径。默认情况下，凭证文件存储在当前用户主目录的 `.aws` 目录中。此属性映射到AWS SDK for .NET中的 `Amazon.AWSConfigs.AWSProfilesLocation` 属性。

region

没有明确指定区域的客户端的默认 AWS 区域 ID。此属性映射到AWS SDK for .NET中的 `Amazon.AWSConfigs.AWSRegion` 属性。

`<aws>` 元素没有父元素。

`<aws>` 元素可包含以下子元素：

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

以下是使用中的 <aws> 元素的示例：

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

dynamoDB

<dynamoDB> 元素表示一组 Amazon DynamoDB 设置。此元素可以包含 conversionSchema 属性，该属性表示在 .NET 和 DynamoDB 对象之间进行转换所用的版本。允许的值包括 V1 和 V2。此属性映射到 AWS SDK for .NET 中的 Amazon.DynamoDBv2.DynamoDBEntryConversion 类。有关更多信息，请参阅 [DynamoDB 系列 - 转换架构](#)。

<dynamoDB> 元素的父级为 <aws> 元素。

<dynamoDB> 元素可以包含 <dynamoDBContext> 子元素。

以下是使用中的 <dynamoDB> 元素的示例：

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

dynamoDBContext

<dynamoDBContext> 元素表示一组 Amazon DynamoDB 上下文特定的设置。此元素可以包含 tableNamePrefix 属性，它表示 DynamoDB 上下文将使用的默认表名前缀 (如果未手动配置此前缀)。此属性从 AWS SDK for .NET 中的 Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix 属性映射到 Amazon.Util.DynamoDBContextConfig.TableNamePrefix 属性。有关更多信息，请参阅 [DynamoDB 开发工具包的增强功能](#)。

<dynamoDBContext> 元素的父级为 <dynamoDB> 元素。

<dynamoDBContext> 元素可包含以下子元素：

- <alias> (一个或多个实例)
- <map> (一个或多个实例)

以下是使用中的 <dynamoDBContext> 元素的示例：

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

ec2

<ec2> 元素表示 Amazon EC2 设置的集合。此元素可以包含 useSignatureVersion4 属性，该属性指定签名版本 4 签名用于所有请求 (true) 还是不用于所有请求 (false，默认值)。此属性从AWS SDK for .NET中的 Amazon.AWSConfigs.EC2Config.UseSignatureVersion4 属性映射到 Amazon.Util.EC2Config.UseSignatureVersion4 属性。

<ec2> 元素的父级为该元素。

<ec2> 元素不包含子元素。

以下是使用中的 <ec2> 元素的示例：

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

日志记录

<logging> 元素表示一组用于响应日志记录和性能指标日志记录的设置。此元素可以包含以下属性：

logMetrics

是否将为所有客户端和配置记录性能指标，如果是，则为 true；否则为 false。此属性从AWS SDK for .NET中的 Amazon.AWSConfigs.LoggingConfig.LogMetrics 属性映射到 Amazon.Util.LoggingConfig.LogMetrics 属性。

logMetricsCustomFormatter

用于日志记录指标的自定义格式化程序的数据类型和程序集名称。此属性从AWS SDK for .NET中的 Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter 属性映射到 Amazon.Util.LoggingConfig.LogMetricsCustomFormatter 属性。

logMetricsFormat

用于表示日志记录指标的格式 (从AWS SDK for .NET中的 `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` 属性映射到 `Amazon.Util.LoggingConfig.LogMetricsFormat` 属性) 。

允许的值包括：

JSON

使用 JSON 格式。

Standard

使用默认格式。

logResponses

记录服务响应的的时间 (从AWS SDK for .NET中的 `Amazon.AWSConfigs.LoggingConfig.LogResponses` 属性映射到 `Amazon.Util.LoggingConfig.LogResponses` 属性) 。

允许的值包括：

Always

始终记录服务响应。

Never

从不记录服务响应。

OnError

仅在出错时记录服务响应。

logTo

记录位置 (从AWS SDK for .NET中的 `Amazon.AWSConfigs.LoggingConfig.LogTo` 属性映射到 `LogTo` 属性) 。

允许的值包括下列一个或多个值：

Log4Net

记录到 log4net。

None

禁用日志记录。

SystemDiagnostics

记录到 System.Diagnostics。

<logging> 元素的父级为 <aws> 元素。

<logging> 元素不包含子元素。

以下是使用中的 <logging> 元素的示例：

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

映射

<map> 元素表示从 .NET 类型到 DynamoDB 表的类型-表映射集合中的单个项（从 AWS SDK for .NET 中的 Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings 属性映射到 TypeMapping 类的实例）。有关更多信息，请参阅 [DynamoDB 开发工具包的增强功能](#)。

此元素可以包含以下属性：

targetTable

映射对应的 DynamoDB 表。此属性映射到 AWS SDK for .NET 中的 Amazon.Util.TypeMapping.TargetTable 属性。

type

映射对应的类型和程序集名称。此属性映射到 AWS SDK for .NET 中的 Amazon.Util.TypeMapping.Type 属性。

<map> 元素的父级为 <dynamoDBContext> 元素。

<map> 元素可包含 <property> 子元素的一个或多个实例。

以下是使用中的 <map> 元素的示例：

```
<map
```

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

property

<property> 元素表示 DynamoDB 属性。(此元素从AWS SDK for .NET中的 AddProperty 方法映射到 Amazon.Util.PropertyConfig 类的实例) 有关更多信息, 请参阅 [DynamoDB 开发工具包的增强功能](#)和 [DynamoDB 属性](#)。

此元素可以包含以下属性:

attribute

属性的属性名, 例如范围键的名称。此属性映射到AWS SDK for .NET中的 Amazon.Util.PropertyConfig.Attribute 属性。

converter

应该用于此属性的转换器的类型。此属性映射到AWS SDK for .NET中的 Amazon.Util.PropertyConfig.Converter 属性。

ignore

是否应忽略关联的属性, 如果忽略, 则为 true; 否则为 false。此属性映射到AWS SDK for .NET中的 Amazon.Util.PropertyConfig.Ignore 属性。

name

属性的名称。此属性映射到AWS SDK for .NET中的 Amazon.Util.PropertyConfig.Name 属性。

version

此属性是否应存储项目版本号, 如果是, 则为 true; 否则为 false。此属性映射到AWS SDK for .NET中的 Amazon.Util.PropertyConfig.Version 属性。

<property> 元素的父级为 <map> 元素。

<property> 元素不包含子元素。

以下是使用中的 <property> 元素的示例:

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

proxy

<proxy> 元素表示用于配置代理以供AWS SDK for .NET使用的设置。此元素可以包含以下属性：

host

代理服务器的主机名或 IP 地址。此属性从AWS SDK for .NET中的 `Amazon.AWSConfigs.ProxyConfig.Host` 属性映射到 `Amazon.Util.ProxyConfig.Host` 属性。

密码

用于对代理服务器进行身份验证的密码。此属性从AWS SDK for .NET 中的 `Amazon.AWSConfigs.ProxyConfig.Password` 属性映射到 `Amazon.Util.ProxyConfig.Password` 属性。

port

代理的端口号。此属性从AWS SDK for .NET中的 `Amazon.AWSConfigs.ProxyConfig.Port` 属性映射到 `Amazon.Util.ProxyConfig.Port` 属性。

username

用于对代理服务器进行身份验证的用户名。此属性从AWS SDK for .NET 中的 `Amazon.AWSConfigs.ProxyConfig.Username` 属性映射到 `Amazon.Util.ProxyConfig.Username` 属性。

<proxy> 元素的父级为 <aws> 元素。

<proxy> 元素不包含子元素。

以下是使用中的 <proxy> 元素的示例：

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


S3

<s3> 元素表示 Amazon S3 设置的集合。此元素可以包含 useSignatureVersion4 属性，该属性指定签名版本 4 签名用于所有请求 (true) 还是不用于所有请求 (false，默认值)。此属性映射到 AWS SDK for .NET 中的 Amazon.AWSConfigs.S3Config.UseSignatureVersion4 属性。

<s3> 元素的父级为 <aws> 元素。

<s3> 元素不包含子元素。

以下是使用中的 <s3> 元素的示例：

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

使用旧版凭证

本部分中的主题提供有关在不使用 AWS IAM Identity Center 的情况下使用长期或短期凭证的信息。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证。而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

Note

本主题中的信息适用于需要手动获取和管理短期或长期凭证的情况。有关短期和长期凭证的更多信息，请参阅《AWS 开发工具包和工具参考指南》中的 [其他身份验证方式](#)。

要了解最佳安全实践，请使用 AWS IAM Identity Center，如 [配置开发工具包身份验证](#) 中所述。

有关凭证的重要警告和指南

有关凭证的警告

- 请勿使用账户的根凭证来访问 AWS 资源。这些凭证可提供不受限的账户访问且难以撤销。
- 请勿在应用程序文件中按字面输入访问密钥或凭证信息。如果您这样做，则在将项目上传到公共存储库或在其他情况下，会有意外暴露凭证的风险。

- 请勿在项目区域中包括含有凭证的文件。
- 请注意，共享 AWS credentials 文件中存储的所有凭证都是以明文形式存储的。

有关安全管理凭证的更多指南

有关如何安全管理 AWS 凭证的一般性讨论，请参阅 [AWS 一般参考](#) 中的 [AWS 安全凭证](#) 和《IAM 用户指南》<https://docs.aws.amazon.com/IAM/latest/UserGuide/> 中的 [安全最佳实践和用例](#)。除了上述讨论内容外，请考虑以下事项：

- 创建其他用户，例如 IAM Identity Center 中的用户，并使用这些用户的凭证，而不是使用您的 AWS 根用户凭证。如有必要，可以撤销其他用户的凭证，或者这些凭证本来就是临时的。此外，您可以对每个用户应用仅允许访问某些资源和操作的策略，从而采取最低权限的立场。
- 对于 Amazon Elastic Container Service (Amazon ECS)，使用 [适用于任务的 IAM 角色](#)。
- 对于在 Amazon EC2 实例上运行的应用程序，使用 [IAM 角色](#)。
- 对可用于组织外部用户的应用程序使用 [临时凭证](#) 或环境变量。

主题

- [使用共享 AWS 凭证文件](#)
- [使用 SDK Store \(仅适用于 Windows \)](#)

使用共享 AWS 凭证文件

(请务必查看 [有关凭证的重要警告和指南](#)。)

为应用程序提供凭证的一种方法是在共享 AWS 凭证文件中创建配置文件，然后将凭证存储在这些配置文件中。此文件可由其它 AWS 开发工具包使用。[AWS CLI](#)、[AWS Tools for Windows PowerShell](#) 以及 [Visual Studio](#)、[JetBrains](#) 和 [VS Code](#) 的 AWS 工具包也可以使用它。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

Note

本主题中的信息适用于需要手动获取和管理短期或长期凭证的情况。有关短期和长期凭证的更多信息，请参阅《AWS 开发工具包和工具参考指南》中的[其他身份验证方式](#)。

要了解最佳安全实践，请使用 AWS IAM Identity Center，如[配置开发工具包身份验证](#)中所述。

一般信息

默认情况下，共享 AWS 凭证文件位于您的主目录中的 `.aws` 目录中，名为 `credentials`；

即 `~/.aws/credentials` (Linux 或 macOS) 或 `%USERPROFILE%\`

`\credentials` (Windows)。有关其它位置的信息，请参阅 [AWS 开发工具包和工具参考指南](#) 中的 [共享文件的位置](#)。另请参阅 [访问应用程序中的凭证和配置文件](#)。

共享 AWS 凭证文件是一个纯文本文件，遵循某种格式。有关 AWS 凭证文件格式的信息，请参阅《AWS 软件开发工具包和工具参考指南》中的 [凭证文件的格式](#)。

用户可通过多种方式管理共享 AWS 凭证文件中的配置文件。

- 使用任意文本编辑器创建和更新共享 AWS 凭证文件。
- 使用 AWS SDK for .NET API 的 [Amazon.Runtime.CredentialManagement](#) 命名空间，如本主题后面所示。
- 使用 [AWS Tools for PowerShell](#) 的命令和程序，以及 [Visual Studio](#)、[JetBrains](#) 和 [VS Code](#) 的 AWS 工具包。
- 使用 [AWS CLI](#) 命令；例如 `aws configure set aws_access_key_id` 和 `aws configure set aws_secret_access_key`。

配置文件管理示例

以下各部分展示了共享 AWS 凭证文件中的配置文件示例。一些示例展示了结果，可以通过前面描述的任何凭证管理方法获得。其它示例展示了如何使用特定方法。

默认配置文件

共享 AWS 凭证文件几乎始终会有一个名为 `default` 的配置文件。如果未定义其它配置文件，AWS SDK for .NET 可以在此处查找凭证。

[default] 配置文件通常如下所示。

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

以编程方式创建配置文件

本示例向您展示了如何以编程方式创建配置文件并将其保存到共享 AWS 凭证文件。它使用 [Amazon.Runtime.CredentialManagement](#) 命名空间的以下类：[CredentialProfileOptions](#)、[CredentialProfile](#) 和 [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

这样的代码通常不会出现在您的应用程序中。如果在应用程序中包含明文密钥，请采取适当的预防措施，确保在代码、网络甚至计算机内存中都看不到明文密钥。

下面是通过本示例创建的配置文件。

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
```

```
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
```

以编程方式更新现有配置文件

本示例向您展示了如何以编程方式更新之前创建的配置文件。它使用 [Amazon.Runtime.CredentialManagement](#) 命名空间的以下类：[CredentialProfile](#) 和 [SharedCredentialsFile](#)。它还使用 [Amazon](#) 命名空间的 [RegionEndpoint](#) 类。

```
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);
    }
}
```

以下是更新的配置文件。

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

Note

您也可以使用其它方法在其它位置设置 AWS 区域。有关更多信息，请参阅[配置 AWS 区域](#)。

使用 SDK Store (仅适用于 Windows)

(请务必查看[重要的警告和指南](#)。)

在 Windows 上，SDK Store 是另一个为 AWS SDK for .NET 应用程序创建配置文件和存储加密凭证的地方。它位于 %USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json。在开发过程中，您可以使用 SDK Store 作为 [共享 AWS 凭证文件](#) 的替代方案。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

Note

本主题中的信息适用于需要手动获取和管理短期或长期凭证的情况。有关短期和长期凭证的更多信息，请参阅《AWS 开发工具包和工具参考指南》中的 [其他身份验证方式](#)。要了解最佳安全实践，请使用 AWS IAM Identity Center，如 [配置开发工具包身份验证](#) 中所述。

一般信息

SDK Store 具有以下优势：

- SDK Store 中的凭证将进行加密，并且 SDK Store 驻留在用户的主目录中。这将限制意外泄露凭证的风险。
- SDK Store 还向 [AWS Tools for Windows PowerShell](#) 和 [AWS Toolkit for Visual Studio](#) 提供凭证。

SDK Store 配置文件针对特定主机上的特定用户。无法将这些配置文件复制到其他主机或其他用户。这表示，您无法在其它主机或开发人员计算机上重用您的开发计算机上的 SDK Store 配置文件。这同时也表示，您不能在生产应用程序中使用 SDK Store 配置文件。

您可通过多种方式管理 SDK Store 中的配置文件。

- 在 [AWS Toolkit for Visual Studio](#) 中使用图形用户界面 (GUI)。
- 使用 AWS SDK for .NET API 的 [Amazon.Runtime.CredentialManagement](#) 命名空间，如本主题后面所示。
- 使用来自 [AWS Tools for Windows PowerShell](#) 的命令；例如，Set-AWSCredential 和 Remove-AWSCredentialProfile。

配置文件管理示例

以下示例说明如何在 SDK Store 中以编程方式创建和更新配置文件。

以编程方式创建配置文件

本示例向您展示了如何以编程方式创建配置文件并将其保存到 SDK Store。它使用 [Amazon.Runtime.CredentialManagement](#) 命名空间的以下类：[CredentialProfileOptions](#)、[CredentialProfile](#) 和 [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

这样的代码通常不会出现在您的应用程序中。如果应用程序中包含明文密钥，请采取适当的预防措施，确保在代码、网络甚至计算机内存中都看不到明文密钥。

下面是通过本示例创建的配置文件。

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

```
"ProfileType" : "AWS",
"DisplayName" : "my_new_profile",
}
```

以编程方式更新现有配置文件

本示例向您展示了如何以编程方式更新之前创建的配置文件。它使用 [Amazon.Runtime.CredentialManagement](#) 命名空间的以下类：[CredentialProfile](#) 和 [NetSDKCredentialsFile](#)。它还使用 [Amazon](#) 命名空间的 [RegionEndpoint](#) 类。

```
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);
    }
}
```

以下是更新的配置文件。

```
"[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

您也可以使用其它方法在其它位置设置 AWS 区域。有关更多信息，请参阅[配置 AWS 区域](#)。

AWS SDK for .NET 的功能

本部分提供了有关在创建应用程序时可能需要考虑的AWS SDK for .NET功能的信息。

请务必先[设置项目](#)。

有关为特定 AWS 服务开发软件及相关代码示例的信息，请参阅[使用 AWS 服务](#)。有关其他代码示例，请参阅 [AWS SDK for .NET 代码示例](#)。

主题

- [适用于 .NET 的 AWS 异步 API](#)
- [重试和超时](#)
- [分页器](#)
- [其他工具](#)

适用于 .NET 的 AWS 异步 API

AWS SDK for .NET使用基于任务的异步模式 (TAP) 进行异步实现。要了解有关 TAP 的更多信息，请参阅 docs.microsoft.com 上的[基于任务的异步模式 \(TAP \)](#)。

本主题概述了在调用 AWS 服务客户端时如何使用 TAP。

AWS SDK for .NET API 中的异步方法是基于 Task 类或 Task<TResult> 类的操作。有关以下这些类的信息，请参阅 docs.microsoft.com : [Task 类](#)、[Task<TResult> 类](#)。

在您的代码中调用这些 API 方法时，必须在使用 async 关键字声明的函数中调用，如以下示例所示。

```
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;
}
```

如前面的代码片段所示，`async` 声明的首选范围是 `Main` 函数。设置此 `async` 范围可确保要求对 AWS 服务客户端的调用都是异步的。如果由于某种原因无法声明 `Main` 为异步，则可以在除 `Main` 之外的函数上使用 `async` 关键字，然后从那里调用 API 方法，如以下示例所示。

```
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;
}
```

请注意使用此模式时 `Main` 所需的特殊 `Task<>` 语法。此外，您必须使用响应的 **Result** 成员来获取数据。

您可以在[快速了解部分](#)（[简单跨平台应用程序](#)和[基于 Windows 的简单应用程序](#)）和[带有指导的代码示例](#)中查看对 AWS 服务客户端进行异步调用的完整示例。

重试和超时

AWS SDK for .NET 让您配置针对 AWS 服务的 HTTP 请求的重试次数和超时值。如果重试和超时的默认值不适用于您的应用程序，您可以针对具体要求调整这些值，但请务必了解这样做将对应用程序的行为带来什么影响。

在确定重试和超时值时，请考虑以下因素：

- 在网络连接降级或无法访问 AWS 服务时，AWS SDK for .NET 和您的应用程序将如何响应？您是希望调用快速失败，还是希望调用代表您不断重试？
- 您的应用程序是必须能够及时响应的面向用户的应用程序或网站，还是更能容忍延迟增加的后台处理任务？
- 应用程序是部署在具有低延迟的可靠网络上，还是部署在具有不可靠连接的远程位置？

重试

概述

AWS SDK for .NET 可以重试因服务器端限制或断开连接导致失败的请求。服务配置类有两个属性可用于指定服务客户端的重试行为。服务配置类继承了抽象的 [Amazon.Runtime 中的这些属性](#)。
ClientConfig [AWS SDK for .NET API 参考](#) 的类别：

- RetryMode 指定三种重试模式之一，这些模式在 [Amazon.Runtime 中定义](#)。RequestRetryMode 枚举。

可以使用 AWS_RETRY_MODE 环境变量或共享 AWS 配置文件中的 retry_mode 设置来控制应用程序的默认值。

- MaxErrorRetry 指定服务客户端级别允许的重试次数；SDK 会在失败并引发异常之前按指定的次数重试该操作。

可以使用 AWS_MAX_ATTEMPTS 环境变量或共享 AWS 配置文件中的 max_attempts 设置来控制应用程序的默认值。

这些属性的详细描述可以在摘要的 [Amazon.Runtime 中找到](#)。ClientConfig [AWS SDK for .NET API 参考](#) 的类别。默认情况下，RetryMode 的每个值对应一个特定的 MaxErrorRetry 值，如下表所示。

RetryMode	Corresponding MaxErrorRetry (Amazon DynamoDB)	Corresponding MaxErrorRetry (all others)
Legacy	10	4
Standard	10	2

RetryMode	Corresponding MaxErrorRetry (Amazon DynamoDB)	Corresponding MaxErrorRetry (all others)
Adaptive (experimental)	10	2

行为

当应用程序启动时

应用程序启动时，RetryMode 和 MaxErrorRetry 的默认值由 SDK 配置。除非您指定其它值，否则在创建服务客户端时将使用这些默认值。

- 如果您的环境中未设置这些属性，则 RetryMode 的默认值将配置为 Legacy，而 MaxErrorRetry 的默认值将配置为上表中的相应值。
- 如果您的环境中设置了重试模式，则该值将用作 RetryMode 的默认值。MaxErrorRetry 的默认值将配置为上表中的相应值，除非您的环境中也设置了最大错误的值（下文将介绍）。
- 如果您的环境中设置了最大错误的值，则该值将用作 MaxErrorRetry 的默认值。Amazon DynamoDB 是该规则的例外；MaxErrorRetry 的默认 DynamoDB 值始终是上表中的值。

当您的应用程序运行时

创建服务客户端时，可以使用 RetryMode 和 MaxErrorRetry 的默认值（如前所述），也可以指定其它值。要指定其他值，请在创建服务客户端时创建并添加服务配置对象，例如 [AmazonDynamodbConfig](#) 或 [AmazonSqsConfig](#)。

一旦为服务客户端创建这些值，便不可更改。

注意事项

在进行重试时，会增加请求的延迟。您应根据应用程序对请求总延迟和错误率的限制来配置重试次数。

超时

AWS SDK for .NET 可让您配置服务客户端级别的请求超时和套接字读取/写入超时值。这些值是在抽象 [Amazon.Runtime](#) 的 [Timeout](#) 和 [ReadWriteTimeout](#) 属性中指定的。 [ClientConfig](#) 班级。这些值作为 AWS 服务客户端 [HttpWebRequest](#) 对象创建的对象 [Timeout](#) 和 [ReadWriteTimeout](#) 属性传递。默认情况下，Timeout 值为 100 秒，ReadWriteTimeout 值为 300 秒。

当您的网络具有高延迟或存在导致操作重试的情况时，使用较长的超时值和较高的重试次数会导致一些开发工具包操作看上去没有响应。

Note

以便携式类库 (PCL) 为目标的版本使用该[HttpClient](#)类而不是[HttpWebRequest](#)类，并且仅支持 [Timeout](#) 属性。AWS SDK for .NET

以下是默认超时值的例外情况。明确设置超时值时将覆盖这些值。

- [Timeout](#) 如果被调用的方法上传直播（例如 [AmazonS3Client](#)），则设置为最大值。[ReadWriteTimeout](#) [PutObjectAsync\(\)](#)，[AmazonS3Client.UploadPartAsync\(\)](#)，[AmazonGlacierClient.UploadArchiveAsync\(\)](#)，依此类推。
- AWS SDK for .NET 该目标 .NET 框架的版本设置为所有 [AmazonS3Client](#) [Timeout](#) 和 [ReadWriteTimeout](#) 对象的最大值。[AmazonGlacierClient](#)
- 针对便携式类库 (PCL) 和 .NET Core 的版本设置 [Timeout](#) 为所有 [AmazonS3Client](#) 和对象的最大值。AWS SDK for .NET [AmazonGlacierClient](#)

示例

以下示例说明如何指定标准重试模式、最多 3 次的重试次数、10 秒超时值以及 10 秒的读取/写入超时值（如果适用）。[AmazonS3Client](#) 构造函数被赋予一个 [AmazonS3Config](#) 对象。

```
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  
    });
```

分页器

某些 AWS 服务会收集和存储大量数据，您可以使用 AWS SDK for .NET 的 API 调用来检索这些数据。如果您要检索的数据量变得太大，无法进行单个 API 调用，则可以通过使用分页将结果分成更易于管理的多个部分。

为了使您能够执行分页，开发工具包中许多服务客户端的请求和响应对象都提供了延续令牌（通常名为 `NextToken`）。其中一些服务客户端还提供了“分页工具”。

分页工具使您能够避免延续令牌的开销，这可能涉及循环、状态变量、多个 API 调用等。使用分页工具时，您可以通过一行代码（`foreach` 循环的声明）从 AWS 服务中检索数据。如果需要多个 API 调用来检索数据，则分页工具会为您处理。

我在哪里可以找到分页工具？

并非所有服务都提供分页工具。确定服务是否为特定 API 提供分页工具的一种方法是在 [AWS SDK for .NET API 参考](#) 中查看服务客户端类的定义。

例如，如果您检查该 [AmazonCloudWatchLogsClient](#) 类的定义，就会看到一个 `Paginatons` 属性。该属性为 Amazon CloudWatch 日志提供了分页器。

分页工具能提供什么好处？

分页工具包含让您能够查看完整响应的属性。它们通常还包含一个或多个属性，使您可以访问响应中最有趣的部分，我们将其称为关键结果。

例如，在前 `AmazonCloudWatchLogsClient` 面提到的中，该 `Paginator` 对象包含一个 `Responses` 属性，其中包含来自 API 调用的完整 [DescribeLogGroupsResponse](#) 对象。此 `Responses` 属性包含日志组集合等内容。

`Paginator` 对象还包含一个名为 `LogGroups` 的关键结果。此属性仅包含响应的日志组部分。有了这个关键结果，您就能在许多情况下减少和简化代码。

同步分页与异步分页

分页工具提供同步和异步分页机制。.NET Framework 4.5（或更高版本）项目中支持同步分页。异步分页可在 .NET 核心项目（.NET Core 3.1、.NET 5 等）中使用。

由于建议使用异步操作和 .NET Core，因此接下来的示例向您展示了异步分页。[分页工具的其他注意事项](#) 中的示例后面显示了有关如何使用同步分页和 .NET Framework 4.5（或更高版本）执行相同任务的信息。

示例

以下示例为您展示了如何使用AWS SDK for .NET显示日志组列表。相比之下，该示例说明如何使用和不使用分页工具执行该操作。在查看完整代码（稍后所示）之前，请考虑以下片段。

获取不带分页器的 CloudWatch 日志组

```
// 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));
```

使用分页器获取 CloudWatch 日志组

```
// 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);
}
```

这两个片段的結果完全相同，因此可以清楚地看到使用分页工具的好处。

Note

在尝试构建和运行完整代码之前，请确保已[设置环境和项目](#)。
你可能还需要 [Microsoft.Bcl.AsyncInterfaces](#) NuGet 打包是因为异步分页器使用该 `IAsyncEnumerable` 接口。

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 包裹：

- [AWSSDK.CloudWatch](#)

编程元素：

- 命名空间 [Amazon。CloudWatch](#)
 - 班级 [AmazonCloudWatchLogsClient](#)
- 命名空间 [Amazon。CloudWatchLogs.Model](#)
 - 班级 [DescribeLogGroupsRequest](#)
 - 班级 [DescribeLogGroupsResponse](#)
 - 班级 [LogGroup](#)

完整代码

```
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}");
    }
}
}
}
```

分页工具的其他注意事项

- 分页工具不能多次使用

如果您需要在代码的多个位置显示特定 AWS 分页工具的结果，则不得多次使用分页工具对象。相反，每次需要时都要创建一个新的分页工具。这个概念如 `DisplayLogGroupsWithPaginators` 方法前面的示例代码中所示。

- 同步分页

同步分页适用于 .NET Framework 4.5 (或更高版本) 的项目。

⚠ Warning

从 2024 年 8 月 15 日起，他们 AWS SDK for .NET 将终止对 .NET Framework 3.5 的支持，并将 .NET Framework 的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化 AWS SDK for .NET](#)。

要查看此内容，请创建一个 .NET Framework 4.5 (或更高版本) 项目，然后将前面的代码复制到该项目中。然后只需将 `await` 关键字从两个 `foreach` 分页工具调用中移除，如以下示例所示。

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

生成并运行该项目，以查看与异步分页相同的结果。

其他工具

以下是一些其它工具，可用于简化 .NET 应用程序的开发、部署和维护工作。

AWS 部署工具

在开发计算机上开发云原生 .NET Core 应用程序后，您可以使用适用于 .NET CLI 的 AWS 部署工具更轻松地将应用程序部署到 AWS。

有关更多信息，请参阅 [将应用程序部署到 AWS](#)。

AWS Message Processing Framework for .NET

这是适用于预览版中功能的预发布文档。本文档随时可能更改。

如果您使用的是亚马逊 SQS、Amazon SNS 或 EventBridge 亚马逊等服务，则可以利用 .NET AWS 的消息处理框架。有关更多信息，请参阅 [AWS Message Processing Framework for .NET](#)。

使用AWS SDK for .NET进行高级身份验证和授权

本部分中的主题提供了有关在AWS SDK for .NET应用程序中使用身份验证和授权的高级技术的信息。

主题

- [通过 AWS SDK for .NET 进行单点登录](#)

通过 AWS SDK for .NET 进行单点登录

AWS IAM Identity Center 是一项基于云的单点登录 (SSO) 服务，可以轻松地集中管理所有 AWS 账户和云应用程序的 SSO 访问。有关完整详细信息，请参阅 [IAM Identity Center 用户指南](#)。

如果您不熟悉开发工具包如何与 IAM Identity Center 交互，请参阅以下信息。

高级互动模式

总体而言，开发工具包与 IAM Identity Center 交互的方式与以下模式类似：

1. 通常通过 [IAM Identity Center 控制台](#) 配置 IAM Identity Center，并邀请 SSO 用户参与。
2. 用户计算机上的共享 AWS config 文件将使用 SSO 信息进行更新。
3. 用户通过 IAM Identity Center 登录，并获得为其配置的 AWS Identity and Access Management (IAM) 权限的短期凭证。这种登录可以通过非 SDK 工具（如 AWS CLI）启动，也可以通过 .NET 应用程序以编程方式启动。
4. 用户继续完成他们的工作。当他们运行其它使用 SSO 的应用程序时，他们无需再次登录即可打开应用程序。

本主题的其余部分提供了设置和使用 AWS IAM Identity Center 的参考信息。它提供了比[配置开发工具包身份验证](#)中基本 SSO 设置更高级的补充信息。如果您不熟悉 AWS 的 SSO，则可能需要先查看该主题以获取基本信息，然后查看以下教程以了解 SSO 的实际操作情况：

- [教程：仅限 .NET 应用程序](#)
- [教程：AWS CLI 和 .NET 应用程序](#)

本主题包含下列部分：

- [先决条件](#)

- [设置 SSO 配置文件](#)
- [生成和使用 SSO 令牌](#)
- [其他资源](#)
- [教程](#)

先决条件

在使用 IAM Identity Center 之前，您必须执行某些任务，例如选择身份来源以及配置相关 AWS 账户和应用程序。有关更多信息，请参阅以下内容：

- 有关这些任务的更多信息，请参阅《IAM Identity Center 用户指南》中的[入门](#)。
- 有关具体任务示例，请参阅本主题末尾的教程列表。但是，在尝试教程之前，请务必查看本主题中的信息。

设置 SSO 配置文件

在相关 AWS 账户中[配置了](#) IAM Identity Center 后，必须将 SSO 的指定配置文件添加到用户的共享 AWS config 文件中。此配置文件用于连接到 [AWS 访问门户](#)，该门户返回已为用户配置的 IAM 权限的短期凭证。

共享 config 文件通常名为 %USERPROFILE%\aws\config (Windows) 和 ~/.aws/config (Linux 和 macOS)。您可以使用首选的文本编辑器为 SSO 添加新的配置文件。或者，您可以使用 `aws configure sso` 命令。有关此命令的更多信息，请参阅《AWS Command Line Interface 用户指南》中的[将 AWS CLI 配置为使用 IAM Identity Center](#)。

新的配置文件类似于以下内容：

```
[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
```

新配置文件的设置定义如下。前两个设置定义了 AWS 访问门户。另外两个设置是一对设置，它们共同定义了已为用户配置的权限。所有四个设置都是必需的。

sso_start_url

指定指向企业的 [AWS 访问门户](#) 的 URL。要找到此值，请打开 [IAM Identity Center 控制台](#)，选择设置，然后找到门户 URL。

sso_region

包含访问门户主机的 AWS 区域。这是您在启用 IAM Identity Center 时选择的区域。它可能与您用于其它任务的区域不同。

有关 AWS 区域及代码的完整列表，请参阅《Amazon Web Services 一般参考》中的 [区域端点](#)。

sso_account_id

通过 AWS Organizations 服务添加 AWS 账户的 ID。要查看可用账户列表，请前往 [IAM Identity Center 控制台](#) 并打开 AWS 账户页面。您为此设置选择的账户 ID 将与您计划为 `sso_role_name` 设置提供的值相对应，如下所示。

sso_role_name

IAM Identity Center 权限集的名称。此权限集定义了通过 IAM Identity Center 向用户授予的权限。

以下过程是查找此设置值的一种方法。

1. 前往 [IAM Identity Center 控制台](#) 并打开 AWS 账户页面。
2. 选择一个账户即可显示其详细信息。您选择的账户将包含您要向其授予 SSO 权限的 SSO 用户或组。
3. 查看分配给该账户的用户和组列表，找到感兴趣的用户或组。您在 `sso_role_name` 设置中指定的权限集是与此用户或组关联的权限集之一。

为该设置指定值时，请使用权限集名称而不是 Amazon 资源名称 (ARN)。

权限集附有 IAM 策略和自定义权限策略。有关更多信息，请参阅《IAM Identity Center 用户指南》中的 [权限集](#)。

生成和使用 SSO 令牌

要使用 SSO，用户必须先生成临时令牌，然后使用该令牌访问相应的 AWS 应用程序和资源。对于 .NET 应用程序，您可以使用以下方法生成和使用这些临时令牌：

- 创建 .NET 应用程序，必要时先生成令牌然后使用该令牌。

- 使用 AWS CLI 生成令牌，然后在 .NET 应用程序中使用该令牌。

这些方法将在以下各节中介绍，并在[教程](#)中进行了演示。

Important

您的应用程序必须引用以下 NuGet 程序包，这样 SSO 解析才能生效：

- AWSSDK.SSO
- AWSSDK.SSO0IDC

未能引用这些程序包将导致运行时系统异常。

仅限 .NET 应用程序

本节介绍如何创建 .NET 应用程序，在必要时生成临时 SSO 令牌，然后使用该令牌。有关此过程的完整教程，请参阅[仅使用 .NET 应用程序的 SSO 教程](#)。

以编程方式生成和使用 SSO 令牌

除了使用 AWS CLI，您还可以通过编程方式生成 SSO 令牌。

为此，您的应用程序会为 SSO 配置文件创建一个 [AWSCredentials](#) 对象，该对象会加载临时凭证（如果有）。然后，您的应用程序必须将 [AWSCredentials](#) 对象转换为 [SSOAWSCredentials](#) 对象并设置一些 [Options](#) 属性，包括用于在必要时提示用户输入登录信息的回调方法。

下面的代码片段中显示了此方法。

Important

您的应用程序必须引用以下 NuGet 程序包，这样 SSO 解析才能生效：

- AWSSDK.SSO
- AWSSDK.SSO0IDC

未能引用这些程序包将导致运行时系统异常。

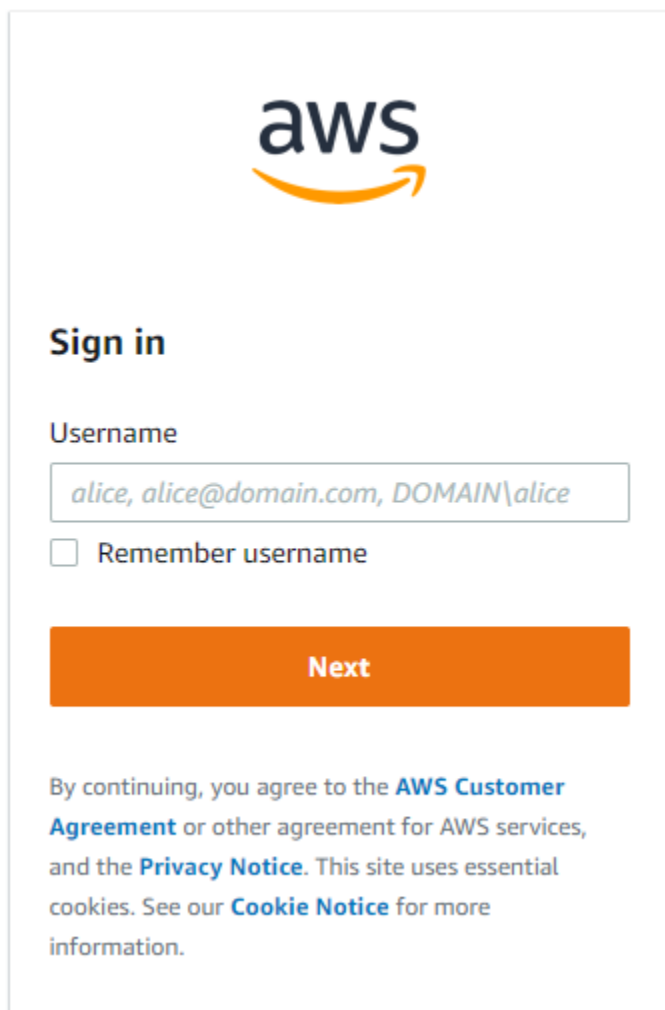
```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    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;
}
```

如果没有相应的 SSO 令牌可用，则会启动默认浏览器窗口并打开相应的登录页面。例如，如果您使用 IAM Identity Center 作为身份来源，则用户会看到类似如下的登录页面：



aws

Sign in

Username

Remember username

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

Note

您为 `SSOAWSCredentials.Options.ClientName` 提供的文本字符串不能有空格。如果字符串确实有空格，则会出现运行时系统异常。

[仅使用 .NET 应用程序的 SSO 教程](#)

AWS CLI 和 .NET 应用程序

本节介绍如何使用 AWS CLI 生成临时 SSO 令牌，以及如何在应用程序中使用该令牌。有关此过程的完整教程，请参阅[使用 AWS CLI 和 .NET 应用程序的 SSO 教程](#)。

使用 AWS CLI 生成 SSO 令牌

除了以编程方式生成临时 SSO 令牌外，您还可以使用 AWS CLI 生成令牌。以下信息将为您演示如何操作。

用户创建启用 SSO 的配置文件后（如[上一节](#)所示），他们将从 AWS CLI 中运行 `aws sso login` 命令。他们必须确保包含带有启用 SSO 的配置文件名称的 `--profile` 参数。如下例所示：

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

如果用户想在当前临时令牌到期后生成新的临时令牌，他们可以再次运行相同的命令。

在 .NET 应用程序中使用生成的 SSO 令牌

以下信息向您展示了如何使用已经生成的临时令牌。

Important

您的应用程序必须引用以下 NuGet 程序包，这样 SSO 解析才能生效：

- AWSSDK.SSO
- AWSSDK.SSO0IDC

未能引用这些程序包将导致运行时系统异常。

您的应用程序为 SSO 配置文件创建一个 [AWSCredentials](#) 对象，该对象会加载之前由 AWS CLI 生成的临时凭证。这与[访问应用程序中的凭证和配置文件](#)中所示的方法类似，其形式如下：

```
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;
}
```

然后将 `AWSCredentials` 对象传递给服务客户端的构造函数。例如：

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

Note

如果您的应用程序已构建为使用 [default] 配置文件进行 SSO，则无需使用 `AWSCredentials` 来加载临时凭证。在这种情况下，应用程序可以创建不带参数的 AWS 服务客户端，类似于“`var client = new AmazonS3Client();`”。

[使用 AWS CLI 和 .NET 应用程序的 SSO 教程](#)

其他资源

如需其它帮助，请参阅以下资源：

- [什么是 IAM Identity Center？](#)
- [将 AWS CLI 配置为使用 IAM Identity Center](#)
- [在AWS Toolkit for Visual Studio中使用 IAM Identity Center 凭证](#)

教程

主题

- [仅使用 .NET 应用程序的 SSO 教程](#)
- [使用 AWS CLI 和 .NET 应用程序的 SSO 教程](#)

仅使用 .NET 应用程序的 SSO 教程

本教程介绍如何为基本应用程序和测试 SSO 用户启用 SSO。它将应用程序配置为以编程方式生成临时 SSO 令牌，而不是[使用 AWS CLI](#)。

本教程向您展示了AWS SDK for .NET中的一小部分 SSO 功能。有关将 IAM Identity Center 与AWS SDK for .NET配合使用的完整详细信息，请参阅带有[背景信息](#)的主题。在该主题中，请特别参见名为[仅限 .NET 应用程序](#)的小节中对此场景的总体描述。

Note

本教程中的几个步骤有助于您配置 AWS Organizations 和 IAM Identity Center 等服务。如果您已经执行了该配置，或者只对代码感兴趣，则可以跳到带有[示例代码](#)的部分。

先决条件

- 请配置您的开发环境（如果尚未配置）。在[安装和配置工具链](#)和[开始使用](#)之类的章节中对此进行了描述。
- 确定或创建至少一个可用于测试 SSO 的 AWS 账户。就本教程而言，名为测试 AWS 账户或者简称为测试账户。
- 确定可以为您测试 SSO 的 SSO 用户。此人将使用 SSO 和您创建的基本应用程序。在本教程中，此人可能是您（开发人员）或其他人。我们还建议采用这样的设置：SSO 用户使用不在您开发环境中的计算机。但是，这并不是必要的。
- SSO 用户的计算机必须安装与您用于设置开发环境的 .NET Framework 兼容的 .NET Framework。

设置 AWS

这一部分演示了如何为本教程设置各种 AWS 服务。

要执行此设置，请先以管理员身份登录测试 AWS 账户。然后执行以下操作：

Amazon S3

前往 [Amazon S3 控制台](#) 并添加一些无关紧要的桶。在本教程的后面部分，SSO 用户将检索这些桶的列表。

AWS IAM

前往 [IAM 控制台](#) 并添加一些 IAM 用户。如果您向 IAM 用户授予权限，请将权限限制为几个无关紧要的只读权限。在本教程的后面部分，SSO 用户将检索这些 IAM 用户的列表。

AWS Organizations

进入 [AWS Organizations 控制台](#) 并启用 Organizations。有关更多信息，请参阅 [AWS Organizations 用户指南](#) 中的 [创建企业](#)。

此操作将测试 AWS 账户作为管理账户添加到组织。如果您还有其它测试账户，则可以邀请他们加入组织，但本教程没有必要这样做。

IAM Identity Center

前往 [IAM Identity Center 控制台](#) 并启用 SSO。如有必要，请执行电子邮件验证。有关更多信息，请参阅 [IAM Identity Center 用户指南](#) 中的 [启用 IAM Identity Center](#)。

然后，执行以下配置。

配置 IAM Identity Center

1. 转到设置页面。查找访问门户 URL 并记录该值供以后在 `sso_start_url` 设置中使用。
2. 在 AWS Management Console 的横幅中，查找启用 SSO 时设置的 AWS 区域。这是 AWS 账户 ID 左侧的下拉菜单。记录区域代码，以便以后在 `sso_region` 设置中使用。此代码将类似于 `us-east-1`。
3. 按如下方式创建 SSO 用户：
 - a. 转到用户页面。
 - b. 选择添加用户，然后输入用户的用户名、电子邮件地址、名字和姓氏。然后选择下一步。
 - c. 在组页面上选择下一步，然后查看信息并选择添加用户。
4. 按如下所示创建组：
 - a. 转至组页面。
 - b. 选择创建组，然后输入组的组名称和描述。
 - c. 在将用户添加到组部分中，选择您之前创建的测试 SSO 用户。选择创建组。
5. 按如下所示创建权限集：
 - a. 转至权限集页面，然后选择创建权限集。
 - b. 在权限集类型下，选择自定义权限集，然后选择下一步。
 - c. 打开内联策略并输入以下策略：

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

```
        "Resource": "*"
    }
  ]
}
```

- d. 在本教程中，输入 `SSOReadOnlyRole` 作为权限集名称。如果需要，可以添加描述，然后选择下一步。
 - e. 检查信息，然后选择创建。
 - f. 记录权限集的名称，以便以后在 `sso_role_name` 设置中使用。
6. 前往 AWS 账户页面，选择您之前添加到组织的 AWS 账户。
 7. 在该页面的概述部分，找到账户 ID 并将其记录下来，以便以后在 `sso_account_id` 设置中使用。
 8. 选择用户和组选项卡，然后选择分配用户或组。
 9. 在分配用户和组页面上，选择组选项卡，选择您之前创建的组，然后选择下一步。
 10. 选择您之前创建的权限集并选择下一步，然后选择提交。配置需要片刻时间。

创建示例应用程序

创建以下应用程序。它们将在 SSO 用户的计算机上运行。

列出 Amazon S3 桶

除 `AWSSDK.S3` 和 `AWSSDK.SecurityToken` 之外，还包括 NuGet 程序包 `AWSSDK.SSO` 和 `AWSSDK.SSO0IDC`。

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

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

namespace 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;
    }
}
}
```

列出 IAM 用户

除 `AWSSDK.IdentityManagement` 和 `AWSSDK.SecurityToken` 之外，还包括 NuGet 程序包 `AWSSDK.SSO` 和 `AWSSDK.SSO0IDC`。

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



```
// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
// AWSSDK.SSOIDC
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 SSOAWSredentials;

    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;
    }
}
}
```

除了显示 Amazon S3 桶和 IAM 用户的列表外，这些应用程序还会显示启用 SSO 的配置文件的用户身份 ARN，在本教程中为 my-sso-profile。

这些应用程序通过在 [SSOAWSCredentials](#) 对象的 [Options](#) 属性中提供回调方法来执行 SSO 登录任务。

SSO 用户指示

让 SSO 用户查看他们的电子邮件并接受 SSO 邀请。系统会提示他们设置密码。这封邮件可能需要几分钟才能送达 SSO 用户的收件箱。

向 SSO 用户提供您之前创建的应用程序。

然后，让 SSO 用户执行以下操作：

1. 如果包含共享 AWS config 文件的文件夹不存在，请创建该文件夹。如果该文件夹确实存在并且有一个名为 .sso 的子文件夹，请删除该子文件夹。

此文件夹的位置通常为 %USERPROFILE%\ .aws (Windows) 以及 ~/.aws (Linux 和 macOS)。

2. 如有必要，在该文件夹中创建共享 AWS config 文件，然后按如下方式向其添加配置文件：

```
[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. 运行 Amazon S3 应用程序。
4. 在最终的 Web 登录页面中，登录。使用邀请消息中的用户名和为回复邮件而创建的密码。
5. 登录完成后，应用程序将显示 S3 桶列表。
6. 运行 IAM 应用程序。该应用程序显示 IAM 用户列表。即使没有进行第二次登录，也是如此。IAM 应用程序使用之前创建的临时令牌。

清除

如果您不想保留在本教程中创建的资源，请将其清理干净。这包括 AWS 资源或开发环境中的资源，例如文件和文件夹。

使用 AWS CLI 和 .NET 应用程序的 SSO 教程

本教程向您展示如何为基本的 .NET 应用程序和测试 SSO 用户启用 SSO。它使用 AWS CLI 生成临时 SSO 令牌，而不是[以编程方式生成](#)。

本教程向您展示了 AWS SDK for .NET 中的一小部分 SSO 功能。有关将 IAM Identity Center 与 AWS SDK for .NET 配合使用的完整详细信息，请参阅带有[背景信息](#)的主题。在该主题中，请特别参见名为[AWS CLI 和 .NET 应用程序](#)的小节中对此场景的总体描述。

Note

本教程中的几个步骤有助于您配置 AWS Organizations 和 IAM Identity Center 等服务。如果您已经执行了这些配置，或者只对代码感兴趣，则可以跳到带有[示例代码](#)的部分。

先决条件

- 请配置您的开发环境（如果尚未配置）。在[安装和配置工具链](#)和[开始使用](#)之类的章节中对此进行了描述。
- 确定或创建至少一个可用于测试 SSO 的 AWS 账户。就本教程而言，名为测试 AWS 账户或者简称为测试账户。
- 确定可以为您测试 SSO 的 SSO 用户。此人将使用 SSO 和您创建的基本应用程序。在本教程中，此人可能是您（开发人员）或其他人。我们还建议采用这样的设置：SSO 用户使用不在您开发环境中的计算机。但是，这并不是必要的。
- SSO 用户的计算机必须安装与您用于设置开发环境的 .NET Framework 兼容的 .NET Framework。
- 确保在 SSO 用户的计算机上[安装了](#) AWS CLI 版本 2。您可以通过在命令提示符或终端中运行 `aws --version` 来检查这一点。

设置 AWS

这一部分演示了如何为本教程设置各种 AWS 服务。

要执行此设置，请先以管理员身份登录测试 AWS 账户。然后执行以下操作：

Amazon S3

前往 [Amazon S3 控制台](#) 并添加一些无关紧要的桶。在本教程的后面部分，SSO 用户将检索这些桶的列表。

AWS IAM

前往 [IAM 控制台](#) 并添加一些 IAM 用户。如果您向 IAM 用户授予权限，请将权限限制为几个无关紧要的只读权限。在本教程的后面部分，SSO 用户将检索这些 IAM 用户的列表。

AWS Organizations

进入 [AWS Organizations 控制台](#) 并启用 Organizations。有关更多信息，请参阅 [AWS Organizations 用户指南](#) 中的 [创建企业](#)。

此操作将测试 AWS 账户作为管理账户添加到组织。如果您还有其它测试账户，则可以邀请他们加入组织，但本教程没有必要这样做。

IAM Identity Center

前往 [IAM Identity Center 控制台](#) 并启用 SSO。如有必要，请执行电子邮件验证。有关更多信息，请参阅 [IAM Identity Center 用户指南](#) 中的 [启用 IAM Identity Center](#)。

然后，执行以下配置。

配置 IAM Identity Center

1. 转到设置页面。查找访问门户 URL 并记录该值供以后在 `sso_start_url` 设置中使用。
2. 在 AWS Management Console 的横幅中，查找启用 SSO 时设置的 AWS 区域。这是 AWS 账户 ID 左侧的下拉菜单。记录区域代码，以便以后在 `sso_region` 设置中使用。此代码将类似于 `us-east-1`。
3. 按如下方式创建 SSO 用户：
 - a. 转到用户页面。
 - b. 选择添加用户，然后输入用户的用户名、电子邮件地址、名字和姓氏。然后选择下一步。
 - c. 在组页面上选择下一步，然后查看信息并选择添加用户。
4. 按如下所示创建组：
 - a. 转至组页面。

- b. 选择创建组，然后输入组的组名称和描述。
 - c. 在将用户添加到组部分中，选择您之前创建的测试 SSO 用户。选择创建组。
5. 按如下所示创建权限集：
- a. 转至权限集页面，然后选择创建权限集。
 - b. 在权限集类型下，选择自定义权限集，然后选择下一步。
 - c. 打开内联策略并输入以下策略：

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

- d. 在本教程中，输入 `SSOReadOnlyRole` 作为权限集名称。如果需要，可以添加描述，然后选择下一步。
 - e. 检查信息，然后选择创建。
 - f. 记录权限集的名称，以便以后在 `sso_role_name` 设置中使用。
6. 前往 AWS 账户页面，选择您之前添加到组织的 AWS 账户。
7. 在该页面的概述部分，找到账户 ID 并将其记录下来，以便以后在 `sso_account_id` 设置中使用。
8. 选择用户和组选项卡，然后选择分配用户或组。
9. 在分配用户和组页面上，选择组选项卡，选择您之前创建的组，然后选择下一步。
10. 选择您之前创建的权限集并选择下一步，然后选择提交。配置需要片刻时间。

创建示例应用程序

创建以下应用程序。它们将在 SSO 用户的计算机上运行。

列出 Amazon S3 桶

除 `AWSSDK.S3` 和 `AWSSDK.SecurityToken` 之外，还包括 NuGet 程序包 `AWSSDK.SSO` 和 `AWSSDK.SSO0IDC`。

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

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

namespace SS0Example.S3.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //   If an active SSO token isn't available, the SSO user should do the
        following:
        //   In a terminal, the SSO user must call "aws sso login --profile my-sso-
        profile".

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

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

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\\nGetting a list of your buckets...");
        }
    }
}
```

```
        var listResponse = await s3Client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
        foreach (S3Bucket b in listResponse.Buckets)
        {
            Console.WriteLine(b.BucketName);
        }
        Console.WriteLine();
    }

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

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

列出 IAM 用户

除 `AWSSDK.IdentityManagement` 和 `AWSSDK.SecurityToken` 之外，还包括 NuGet 程序包 `AWSSDK.SSO` 和 `AWSSDK.SSO0IDC`。

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

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
AWSSDK.SSO0IDC
```



```
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //   If an active SSO token isn't available, the SSO user should do the
following:
        //   In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

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

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

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

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

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

除了显示 Amazon S3 桶和 IAM 用户的列表外，这些应用程序还会显示启用 SSO 的配置文件的用户身份 ARN，在本教程中为 my-sso-profile。

SSO 用户指示

让 SSO 用户查看他们的电子邮件并接受 SSO 邀请。系统会提示他们设置密码。这封邮件可能需要几分钟才能送达 SSO 用户的收件箱。

向 SSO 用户提供您之前创建的应用程序。

然后，让 SSO 用户执行以下操作：

1. 如果包含共享 AWS config 文件的文件夹不存在，请创建该文件夹。如果该文件夹确实存在并且有一个名为 .sso 的子文件夹，请删除该子文件夹。

此文件夹的位置通常为 %USERPROFILE%\ .aws (Windows) 以及 ~/.aws (Linux 和 macOS)。

2. 如有必要，在该文件夹中创建共享 AWS config 文件，然后按如下方式向其添加配置文件：

```
[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. 运行 Amazon S3 应用程序。出现运行时系统异常。
4. 运行以下 AWS CLI 命令：

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

5. 在最终的 Web 登录页面中，登录。使用邀请消息中的用户名和为回复邮件而创建的密码。
6. 再次运行 Amazon S3 应用程序。现在，应用程序显示 S3 桶的列表。
7. 运行 IAM 应用程序。该应用程序显示 IAM 用户列表。即使没有进行第二次登录，也是如此。IAM 应用程序使用之前创建的临时令牌。

清除

如果您不想保留在本教程中创建的资源，请将其清理干净。这包括 AWS 资源或开发环境中的资源，例如文件和文件夹。

将应用程序部署到 AWS

在开发计算机上开发云原生 .NET Core 应用程序或服务后，您需要将其部署到 AWS。您可以使用 AWS Management Console 或某些服务（如 AWS CloudFormation 或 AWS Cloud Development Kit (AWS CDK)）来完成此操作。您也可以使用因部署而创建的 AWS 工具。通过使用这些工具，您可以执行以下操作。

从 .NET CLI 部署

您可以使用以下适用于 .NET CLI 的 AWS 工具将应用程序部署到 AWS：

- [适用于 .NET CLI 的 AWS 部署工具](#) - 支持部署到 [AWS App Runner](#)、[Amazon Elastic Container Service \(Amazon ECS\)](#) 和 [AWS Elastic Beanstalk](#)。
- [适用于 .NET CLI 的 AWS Lambda 工具](#) - 支持 AWS Lambda 项目的部署。

通过 IDE 工具包部署

您可以使用 AWS 工具包直接从您选择的 IDE 部署应用程序：

- [AWS Toolkit for Visual Studio](#)

Note

该工具包中的“发布到 AWS”功能与适用于 .NET CLI 的 AWS 部署工具具有相同的功能。要了解更多信息，请转到《AWS Toolkit for Visual Studio 用户指南》中的[发布到 AWS](#)。

- [AWS Toolkit for JetBrains](#)

请参见[使用 AWS 无服务器应用程序](#)和[使用 AWS App Runner](#)。

- [AWS Toolkit for VS Code](#)

请参阅[使用无服务器应用程序](#)和[使用 AWS App Runner](#)

- [AWS Toolkit for Azure DevOps](#)

用例

以下各节包含某些类型应用程序的用例场景，包括有关如何使用 .NET CLI 部署这些应用程序的信息。

- [ASP.NET Core 应用程序](#)
- [.NET Console 应用程序](#)
- [Blazor 应用程序 WebAssembly](#)
- [AWS Lambda 项目](#)

ASP.NET Core 应用程序

适用于 .NET CLI 的 [AWS 部署工具](#) 有助于您部署 ASP.NET 应用程序并指导您完成部署过程。它是 .NET CLI 的交互式工具，有助于部署 .NET 应用程序，几乎不需要具备什么 AWS 知识。

部署工具具有以下功能：

- 针对您的应用程序的计算建议 - 获取计算建议，并了解哪种 AWS 计算最适合您的应用程序。
- 生成 Dockerfile - 如果需要，该工具会生成 Dockerfile，或者使用现有的 Dockerfile。
- 自动打包和部署 - 该工具可构建部署构件，使用生成的 AWS CDK 部署项目配置基础设施，并将您的应用程序部署到所选的 AWS 计算中。
- 可重复且可共享的部署 - 您可以生成和修改 AWS CDK 部署项目以适应您的特定使用案例。您还可以对项目进行版本控制并与团队共享，以实现可重复的部署。
- 有助于学习 AWS CDK for .NET - 该工具有助于您逐步学习其构建所基于的底层 AWS 工具，例如 AWS CDK。

[AWS 部署工具](#) 支持将 ASP.NET Core 应用程序部署到以下 AWS 服务：

- 使用 [AWS Fargate](#) 的 [Amazon ECS 服务](#) - 支持将 Web 应用程序部署到 Amazon Elastic Container Service (Amazon ECS)，计算能力由 AWS Fargate 无服务器计算引擎管理。
- [AWS App Runner](#) - 支持部署到完全托管的服务，使开发人员可以轻松地大规模部署容器化 Web 应用程序和 API。无需事先具备基础设施经验。
- [AWS Elastic Beanstalk](#) - 支持部署到一项服务，使开发人员可以轻松地将 Web 应用程序和 API 部署到完全托管的环境。无需事先具备基础设施经验。

要了解更多信息，请参阅[工具概述](#)。要从那里开始，请导航至文档、入门，然后选择[如何安装](#)来获取安装说明。

.NET Console 应用程序

适用于 .NET CLI 的 [AWS 部署工具](#) 有助于您部署 .NET Console 应用程序作为服务，或者部署计划任务作为容器映像，并指导您完成部署过程。如果您的应用程序没有 Dockerfile，则该工具会自动生成。否则，将使用现有的 Dockerfile。

部署工具具有以下功能：

- 针对您的应用程序的计算建议 - 获取计算建议，并了解哪种 AWS 计算最适合您的应用程序。
- 生成 Dockerfile - 如果需要，该工具会生成 Dockerfile，或者使用现有的 Dockerfile。
- 自动打包和部署 - 该工具可构建部署构件，使用生成的 AWS CDK 部署项目配置基础设施，并将您的应用程序部署到所选的 AWS 计算中。
- 可重复且可共享的部署 - 您可以生成和修改 AWS CDK 部署项目以适应您的特定使用案例。您还可以对项目进行版本控制并与团队共享，以实现可重复的部署。
- 有助于学习 AWS CDK for .NET - 该工具有助于您逐步学习其构建所基于的底层 AWS 工具，例如 AWS CDK。

[AWS 部署工具](#) 支持将 .NET Console 应用程序部署到以下 AWS 服务：

- 使用 [AWS Fargate](#) 的 [Amazon ECS 服务](#) - 支持将 .NET 应用程序作为一项服务（例如后台处理器）部署到 Amazon Elastic Container Service（Amazon ECS），计算能力由 AWS Fargate 无服务器计算引擎管理。
- [Amazon ECS 计划任务](#) 使用 [AWS Fargate](#) - 支持将 .NET 应用程序作为计划任务（例如 end-of-day 进程）部署到 Amazon ECS，计算能力由 AWS Fargate 无服务器计算引擎管理。

要了解更多信息，请参阅[工具概述](#)。要从那里开始，请导航至文档、入门，然后选择[如何安装](#)来获取安装说明。

Blazor 应用程序 WebAssembly

适用于 .NET CLI 的 [AWS 部署工具](#) 可帮助您在亚马逊 S3 中托管 Blazor WebAssembly 应用程序，使用亚马逊 CloudFront 进行内容网络交付。您的应用程序已部署到用于 Web 托管的 S3 桶。该工具创建和配置 S3 桶，然后将您的 Blazor 应用程序上传到该桶。

部署工具具有以下功能：

- 自动打包和部署 - 该工具可构建部署构件，使用生成的 AWS CDK 部署项目配置基础设施，并将您的应用程序部署到所选的 AWS 计算中。
- 可重复且可共享的部署 - 您可以生成和修改 AWS CDK 部署项目以适应您的特定使用案例。您还可以对项目进行版本控制并与团队共享，以实现可重复的部署。
- 有助于学习 AWS CDK for .NET - 该工具有助于您逐步学习其构建所基于的底层 AWS 工具，例如 AWS CDK。

要了解更多信息，请参阅[工具概述](#)。要从那里开始，请导航至文档、入门，然后选择[如何安装](#)来获取安装说明。

AWS Lambda 项目

AWS Lambda 是一项计算服务，可使您无需预配置或管理服务器即可运行代码。它在可用性高的计算基础设施上运行您的代码，并执行计算资源的所有管理工作。有关 Lambda 的更多信息，请参阅 AWS Lambda 开发人员指南中的[什么是 AWS Lambda？](#)

您可以使用 .NET 命令行界面 (CLI) 部署 Lambda 函数。

主题

- [先决条件](#)
- [可用的 Lambda 命令](#)
- [部署步骤](#)

先决条件

在开始使用 .NET CLI 部署 Lambda 函数之前，您必须满足以下先决条件：

- 确认您已安装 .NET CLI。例如：`dotnet --version`。如果需要，请前往 <https://dotnet.microsoft.com/download> 进行安装。
- 设置 .NET CLI 以使用 Lambda。有关如何执行此操作的说明，请参阅《AWS Lambda 开发人员指南》中的 [.NET Core CLI](#)。在该过程中，以下是部署命令：

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

如果您不确定如何为此练习创建 IAM 角色，请不要包括 `--function-role role` 部分。该工具将有助于您创建新角色。

可用的 Lambda 命令

要列出可通过 .NET CLI 使用的 Lambda 命令，请打开命令提示符或终端并输入 `dotnet lambda --help`。该命令输出将与以下内容类似：

```
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>
```

输出列出了当前可用的所有命令。

部署步骤

以下说明假设您已创建 AWS Lambda .NET 项目。出于该过程的目的，该项目被命名为 `DotNetCoreLambdaTest`。

1. 打开命令提示符或终端，并导航到包含您的 .NET Lambda 项目文件的文件夹。
2. 输入 `dotnet lambda deploy-function`。
3. 当系统提示时，输入 AWS 区域（Lambda 函数将部署到的区域）。
4. 当系统提示时，输入要部署的函数的名称，例如 `DotNetCoreLambdaTest`。它可以是您 AWS 账户中已经存在的函数的名称，也可以是尚未部署的函数的名称。
5. 当系统提示时，选择或创建 Lambda 将在执行函数时代入的 IAM 角色。

成功完成后，将显示消息新 Lambda 函数已创建。

```
Executing publish command
```



```
...  
(etc.)  
New Lambda function created
```

如果您部署的函数在您的账户中已存在，则部署函数仅要求提供 AWS 区域（如有必要）。在这种情况下，命令输出以 `Updating code for existing function` 结尾。

在部署 Lambda 函数后，便可使用该函数。有关更多信息，请参阅[如何使用 AWS Lambda 的示例](#)。

Lambda 会自动通过 Amazon CloudWatch 为您监控 Lambda 函数并报告指标。要监控您的 Lambda 函数并对其进行故障排除，请参阅[对 Lambda 应用程序进行监控和故障排除](#)。

为AWS SDK for .NET迁移您的项目

本部分提供有关可能适用于您的迁移任务的信息，以及有关如何执行这些任务的说明。

主题

- [AWS SDK for .NET中的新增内容](#)
- [AWS SDK for .NET支持的平台](#)
- [迁移到AWS SDK for .NET的版本 3](#)
- [迁移到AWS SDK for .NET 3.5 版](#)
- [迁移到AWS SDK for .NET 3.7 版](#)
- [从 .NET Standard 1.3 迁移](#)

AWS SDK for .NET中的新增内容

有关与... 相关的新开发的高级信息，请参阅产品页面 <https://aws.amazon.com/sdk-for-net/> AWS SDK for .NET。

以下是AWS SDK for .NET中的新增内容。

2024 年 2 月 18 日：即将对.NET 框架支持的更改

从 2024 年 8 月 15 日起，他们AWS SDK for .NET将终止对.NET Framework 3.5 的支持，并将.NET 框架的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化AWS SDK for .NET](#)。

2023-07-17：AWS Lambda 注释框架已正式发布

通过使用 C# 源代码生成器技术，[AWS Lambda 注释框架](#)使 .NET 开发人员在使用 C# 编写 Lambda 函数的体验变得更加自然。现已正式发布。

2023-07-15：DynamoDB 的 Distributed Cache Provider 已发布预览版

这是适用于预览版中功能的预发布文档。本文档随时可能更改。

Distributed Cache Provider 库允许将 Amazon DynamoDB 用作 ASP.NET Core 分布式缓存框架的存储。[有关更多信息，请参阅博客文章介绍适用于 DynamoDB 的 AWS .NET 分布式缓存提供商 \(预览版\) 和存储库。GitHub](#)

2022-07-13 : AWS 部署工具已发布

AWS 部署工具已发布。该工具是适用于 .NET CLI 的交互式工具，AWS Toolkit for Visual Studio 有助于部署 .NET 应用程序，且不需要什么 AWS 知识，需要的操作和命令也极少。有关更多信息，请参阅[将应用程序部署到 AWS](#)。

2020-08-24 : 开发工具包 3.5 版已经发布

- 通过将对所有开发工具包的非 Framework 版本的支持转换为 .NET Standard 2.0，进一步标准化了 .NET 体验。请参阅[迁移到 3.5 版](#)了解更多信息。
- 为许多服务客户端添加了分页工具，这会让 API 结果的分页更加方便。有关更多信息，请参阅[分页器](#)。

AWS SDK for .NET 支持的平台

AWS SDK for .NET 面向不同平台为开发人员提供了不同程序集组。但是，并非所有这些平台上的开发工具包功能均相同。本主题介绍了各个平台中的支持差异。

.NET 内核

AWS SDK for .NET 支持为 .NET Core (.NET Core 3.1、.NET 5、.NET 6 等) 编写的应用程序。AWS 服务客户端仅支持 .NET Core 中的异步调用模式。这还会影响到多种在 Amazon S3 TransferUtility 等服务客户端上构建的高级别抽象，这些客户端只支持 .NET Core 环境中的异步调用。

.NET Standard 2.0

非 Framework 版本的 AWS SDK for .NET 符合 [.NET Standard 2.0](#)。AWS SDK for .NET 仅为根据 .NET Standard 编写的应用程序提供异步方法。

.NET Framework 4.5

Warning

从 2024 年 8 月 15 日起，他们 AWS SDK for .NET 将终止对 .NET Framework 3.5 的支持，并将 .NET 框架的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化 AWS SDK for .NET](#)。

此版本的AWS SDK for .NET根据 .NET Framework 4.5 进行编译，并在 .NET 4.0 运行时系统中运行。AWS 服务客户端支持同步和异步调用模式，并使用 [C# 5.0](#) 中引入的 [async](#) 和 [await](#) 关键字。

.NET Framework 3.5

Warning

从 2024 年 8 月 15 日起，他们AWS SDK for .NET将终止对.NET Framework 3.5 的支持，并将.NET 框架的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化AWS SDK for .NET](#)。

此版本的AWS SDK for .NET根据 .NET Framework 3.5 进行编译，并在 .NET 2.0 或 .NET 4.0 运行时系统中运行。AWS 服务客户端支持同步和异步调用模式，并使用旧版的 Begin 和 End 模式。

Note

在由针对版本 2.0 的 CLR 构建的应用程序使用时，AWS SDK for .NET不符合联邦信息处理标准 (FIPS)。有关如何在该环境中替代符合 FIPS 的实现的信息，请参阅[微软博客](#)和 Security.cryptography.dll 中的 [CLR 安全](#)团队的 HMACSHA256 类 (hmacsha256CNG)。 [CryptoConfig](#)

便携式类库和 Xamarin

AWS SDK for .NET还包含可移植类库实施。便携式类库实施可针对多种平台，包括 Universal Windows Platform (UWP) 以及 iOS 和 Android 上的 Xamarin。有关更多详细信息，请参阅[适用于 .NET 的移动开发工具包和 Xamarin](#)。AWS 服务客户端仅支持异步调用模式。

Unity 支持

有关 Unity 支持的信息，请参阅 [Unity 支持的特殊注意事项](#)。

更多信息

[迁移到AWS SDK for .NET 3.5 版](#)

迁移到AWS SDK for .NET的版本 3

本主题介绍了版本 3 的AWS SDK for .NET中的更改以及如何将您的代码迁移到此版本的开发工具包。

关于AWS SDK for .NET版本

AWS SDK for .NET (最初于 2009 年 11 月发布) 专为 .NET Framework 2.0 而设计。自发布以来, .NET 已通过 .NET Framework 4.0 和 .NET Framework 4.5 进行了改进, 并且已增加了新的目标平台: WinRT 和 Windows Phone。

AWS SDK for .NET版本 2 已更新, 可以利用 .NET 平台的新功能并且面向 WinRT 和 Windows Phone。

AWS SDK for .NET版本 3 已进行更新, 使程序集实现模块化。

面向开发工具包的架构重新设计

整个版本 3 的AWS SDK for .NET已进行重新设计来实现模块化。现在, 每项服务均在各自的程序集中而非某个全局程序集中实现。您不再需要将整个AWS SDK for .NET添加到应用程序。现在, 您可以仅为应用程序使用的 AWS 服务添加程序集。

重大更改

以下部分介绍了对版本 3 的AWS SDK for .NET的更改。

删除了 AWSClientFactory

删除了 Amazon.AWSClientFactory 类。现在, 要创建服务客户端, 请使用服务客户端的构造函数。例如, 要创建 AmazonEC2Client, 请使用:

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

删除了 Amazon.Runtime.AssumeRoleAWSCredentials

删除了 Amazon.Runtime.AssumeRoleAWSCredentials 类, 因为它位于核心命名空间中但依赖于 AWS Security Token Service, 并且开发工具包中已将其弃用一段时间。请改用 Amazon.SecurityToken.AssumeRoleAWSCredentials 类。

从 S3Link 中删除了 SetACL 方法

S3Link 类是 Amazon.DynamoDBv2 程序包的一部分，用于将作为 DynamoDB 项中的参考的对象存储在 Amazon S3 中。虽然这是一项有用的功能，但我们不需要为 DynamoDB 创建 Amazon.S3 程序包的编译依赖项。因此，我们从 S3Link 类简化了公开的 Amazon.S3 方法，并将 SetACL 方法替换为 MakeS3ObjectPublic 方法。要更好地控制对象的访问控制列表 (ACL)，请直接使用 Amazon.S3 程序包。

删除过时的结果类

对于 AWS SDK for .NET 中的大多数服务，操作将返回一个响应对象，该对象包含操作的元数据，例如请求 ID 和结果对象。拥有一个单独的响应和结果类是多余的，并且会让开发人员编写额外的代码。在版本 2 的 AWS SDK for .NET 中，我们将结果类中的所有信息放入响应类中。我们还将结果类标记为过时以阻止对结果类的使用。在版本 3 的 AWS SDK for .NET 中，我们删除了这些过时的结果类，这样有助于减小开发工具包的大小。

AWS Config 部分更改

可以通过 App.config 或 Web.config 文件执行 AWS SDK for .NET 的高级配置。您可以通过与以下内容类似的 <aws> 配置部分 (将引用开发工具包程序集名称) 来执行此操作。

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

在版本 3 的 AWS SDK for .NET 中，AWSSDK 程序集不再存在。我们将常见代码放入 AWSSDK.Core 程序集中。因此，您需要将对 App.config 或 Web.config 文件中的 AWSSDK 程序集的引用更改为对 AWSSDK.Core 程序集的引用，如下所示。

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

```
</aws>  
</configuration>
```

您也可以使用 `Amazon.AWSConfigs` 类操作配置设置。在 AWS SDK for .NET 的版本 3 中，我们已将 DynamoDB 的配置设置从 `Amazon.AWSConfigs` 类更改为 `Amazon.AWSConfigsDynamoDB` 类。

迁移到 AWS SDK for .NET 3.5 版

AWS SDK for .NET 的版本 3.5 通过将对所有开发工具包的非 Framework 变体的支持转换为 [.NET Standard 2.0](#)，进一步标准化了 .NET 体验。根据您的环境和代码库，要利用 3.5 版功能，您可能需要执行某些迁移工作。

本主题介绍版本 3.5 中的更改以及从版本 3 迁移环境或代码可能需要执行的工作。

3.5 版的更改内容

以下内容介绍 AWS SDK for .NET 版本 3.5 中已更改或尚未更改的内容。

.NET Framework 和 .NET Core

对 .NET Framework 和 .NET Core 的支持没有更改。

Xamarin

(新的和现有的) Xamarin 项目必须指向 .NET Standard 2.0。请参阅 [Xamarin.Forms 中的 .NET Standard 2.0 支持](#) 和 [.NET 实现支持](#)。

Unity

Unity 应用程序必须使用 Unity 2018.1 或更高版本指向 .NET Standard 2.0 或 .NET 4.x 配置文件。有关更多信息，请参阅 [.NET 配置文件支持](#)。此外，如果您使用 IL2CPP 来构建，则必须通过添加 link.xml 文件来禁用代码剥离，如从 [Unity、Xamarin 或 UWP 中引用 AWS SDK for .NET \(Standard 2.0 \)](#) 中所述。将代码移植到推荐的代码库之一后，Unity 应用程序可以访问开发工具包提供的所有服务。

由于 Unity 支持 .NET Standard 2.0，因此开发工具包版本 3.5 的 `AWSSDK.core` 程序包不再具有 Unity 特定的代码，包括一些更高级别的功能。为了提供更好的转换，所有 legacy 的 Unity 代码都可以在 [aws/aws-sdk-unity-net](#) GitHub 存储库中供参考。如果您发现缺少影响将 AWS 与 Unity 结合使用的功能，您可以通过 <https://github.com/aws/dotnet/issues> 提交功能请求。

另请参阅[Unity 支持的特殊注意事项](#)。

通用 Windows 平台 (UWP)

将您的 UWP 应用程序指向[版本 16299 或更高版本](#) (秋季创建者更新, 版本 1709, 2017 年 10 月发布)。

Windows Phone 和 Silverlight

AWS SDK for .NET 的版本 3.5 不支持这些平台, 因为 Microsoft 不再积极开发它们。有关更多信息, 请参阅下列内容:

- [Windows 10 Mobile 终止支持](#)
- [Silverlight 终止支持](#)

传统的便携式类库 (基于配置文件的 PCL)

考虑将您的库重定向到 .NET 标准。有关更多信息, 请参阅 Microsoft 提供的[可移植类库的比较](#)。

Amazon Cognito Sync Manager 和 Amazon Mobile Analytics Manager

AWS SDK for .NET 3.5 版中删除了便于使用 Amazon Cognito Sync 和 Amazon Mobile Analytics 的高级抽象。AWS AppSync 是 Amazon Cognito Sync 的首选替代品。Amazon Pinpoint 是 Amazon Mobile Analytics 的首选替代品。

如果您的代码受到缺乏 AWS AppSync 和 Amazon Pinpoint 的更高级别库代码的影响, 您可以记录您对以下 GitHub 问题之一或这两个问题的兴趣: <https://github.com/aws/dotnet/issues/20> 和 <https://github.com/aws/dotnet/issues/19>。您还可以从以下 GitHub 存储库获取 Amazon Cognito Sync Manager 和 Amazon Mobile Analytics Manager 的库: [aws/amazon-cognito-sync-manager-net](https://github.com/aws/amazon-cognito-sync-manager-net) 和 [aws/aws-mobile-analytics-manager-net](https://github.com/aws/aws-mobile-analytics-manager-net)。

迁移同步代码

AWS SDK for .NET 3.5 版同时支持 .NET Framework 和 .NET Standard (通过 .NET Core 3.1、.NET 5 等 .NET Core 版本)。符合 .NET Standard 的开发工具包版本仅提供异步方法, 因此, 如果您想利用 .NET Standard, 则必须更改同步代码以使其异步运行。

以下代码片段展示如何将同步代码更改为异步代码。这些代码片段中的代码用于显示 Amazon S3 桶的数量。

原始代码调用 [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}");
```

要使用此开发工具包的 3.5 版本，请改为调用 [ListBucketsAsync](#)。

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

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

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

迁移到AWS SDK for .NET 3.7 版

从 3.7 版开始，AWS SDK for .NET 不再支持 .NET Standard 1.3。

有关从 .NET Standard 1.3 迁移的信息，请参阅[从 .NET Standard 1.3 迁移](#)。

从 .NET Standard 1.3 迁移

2019 年 6 月 27 日，Microsoft [终止了](#)对 .NET Core 1.0 和 .NET Core 1.1 版本的支持。宣布此消息之后，AWS 已于 2020 年 12 月 31 日终止在 AWS SDK for .NET 上支持 .NET Standard 1.3。

AWS 后续继续在 AWS SDK for .NET 上针对 .NET Standard 1.3 提供服务更新和安全修复，直至 2020 年 10 月 1 日。在此日期之后，.NET Standard 1.3 目标进入维护模式，这意味着不会发布新的更新；AWS 仅应用了关键错误修复和安全补丁。

在 2020 年 12 月 31 日，对 AWS SDK for .NET 上的 .NET Standard 1.3 的支持结束。在此日期之后，未应用错误修复或安全补丁。使用该目标构建的构件仍可在 NuGet 上下载。

您需要了解的内容

- 如果您正在运行使用 .NET Framework 的应用程序，则不会受到影响。
- 如果您正在运行使用 .NET Core 2.0 或更高版本的应用程序，则不会受到影响。
- 如果您正在运行使用 .NET Core 1.0 或 .NET Core 1.1 的应用程序，请按照 [Microsoft 迁移说明](#) 将应用程序迁移到 .NET Core 的较新版本。我们建议至少使用 .NET Core 3.1。
- 如果您正在运行目前无法升级的关键业务应用程序，则可以继续使用当前版本的 AWS SDK for .NET。

如果您有疑问或疑虑，请[联系 AWS Support](#)。

在 AWS SDK for .NET 中使用 AWS 服务

以下各部分包含示例、教程、任务和指南，向您展示如何使用AWS SDK for .NET来利用 AWS 服务。这些示例和教程依赖于 AWS SDK for .NET 提供的 API。要查看 API 中有哪些类和方法可用，请参阅[AWS SDK for .NET API 参考](#)。

如果您不熟悉AWS SDK for .NET，则可能需要先查看[快速了解](#)主题。其中提供了对 SDK 的简单介绍。

您可以在 [AWS 代码示例存储库](#) 和 GitHub 上的 [awslabs repository](#) 中找到更多代码示例。

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

主题

- [带有适用于AWS SDK for .NET的引导的代码示例](#)
- [将 AWS Lambda 用于计算服务](#)
- [AWS SDK for .NET 的高级库和框架](#)
- [对 AWS OpsWorks 进行编程以使用堆栈和应用程序](#)
- [对其它 AWS 服务和配置的支持](#)

带有适用于AWS SDK for .NET的引导的代码示例

以下各节包含代码示例，并提供示例指导。其将有助于您了解如何使用AWS SDK for .NET来利用 AWS 服务。

如果您不熟悉AWS SDK for .NET，则可能需要先查看[快速了解](#)主题。其中提供了对 SDK 的简单介绍。

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

主题

- [使用AWS SDK for .NET访问 AWS CloudFormation](#)
- [使用 Amazon Cognito 验证用户身份](#)
- [使用 Amazon DynamoDB NoSQL 数据库](#)
- [使用 Amazon EC2](#)
- [使用AWS SDK for .NET访问 AWS Identity and Access Management \(IAM \)](#)

- [使用 Amazon Simple Storage Service Internet 存储](#)
- [使用 Amazon Simple Notification Service 从云端发送通知](#)
- [使用 Amazon SQS 发送消息](#)

使用AWS SDK for .NET访问 AWS CloudFormation

AWS SDK for .NET支持 [AWS CloudFormation](#)，它预先反复创建和预置 AWS 基础设施部署。

API

AWS SDK for .NET为 AWS CloudFormation 客户端提供 API。这些 API 使您能够使用模板和堆栈等 AWS CloudFormation 功能。本节包含少量示例，向您展示使用这些 API 时可以遵循的模式。要查看完整的 API 集，请参阅[AWS SDK for .NET API 参考](#)（并滚动至“Amazon.CloudFormation”）。

AWS CloudFormation API 由 [AWSSDK.CloudFormation](#) 程序包提供。

先决条件

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

主题

主题

- [使用 AWS CloudFormation 列出 AWS 资源](#)

使用 AWS CloudFormation 列出 AWS 资源

此示例向您展示如何使用AWS SDK for .NET列出 AWS CloudFormation 堆栈中的资源。该示例使用低级别 API。该应用程序不带任何参数，只是收集用户凭证可以访问的所有堆栈的信息，然后显示有关这些堆栈的信息。

SDK 参考

NuGet 程序包：

- [AWSSDK.CloudFormation](#)

编程元素：

- 命名空间 [Amazon.CloudFormation](#)
 - 类 [AmazonCloudFormationClient](#)
- 命名空间 [Amazon.CloudFormation.Model](#)
 - 类 [DescribeStackResourcesRequest](#)
 - 类 [DescribeStackResourcesResponse](#)
 - 类 [DescribeStacksResponse](#)
 - 类 [Stack](#)
 - 类 [StackResource](#)
 - 类 [Tag](#)

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

namespace CFNListResources
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Create the CloudFormation client
            var cfnClient = new AmazonCloudFormationClient();

            // List the resources for each stack
            await ListResources(cfnClient, await cfnClient.DescribeStacksAsync());
        }

        //
        // Method to list stack resources and other information
        private static async Task ListResources(
            IAmazonCloudFormation cfnClient, DescribeStacksResponse responseDescribeStacks)
        {
            Console.WriteLine("Getting CloudFormation stack information...");
        }
    }
}
```

```
foreach (Stack stack in responseDescribeStacks.Stacks)
{
    // Basic information for each stack
    Console.WriteLine("\n-----");
    Console.WriteLine($"Stack: {stack.StackName}");
    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
    Console.WriteLine($"  Created: {stack.CreationTime}");

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

    // The resources of each stack
    DescribeStackResourcesResponse responseDescribeResources =
        await cfnClient.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-----");
}
}
```

使用 Amazon Cognito 验证用户身份

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

使用 Amazon Cognito Identity，您可以为用户创建唯一的身份并对其进行身份验证，以实现对其对 AWS 资源 (如 Amazon S3 或 Amazon DynamoDB) 的安全访问。Amazon Cognito Identity 支持公共身份提供商 (如 Amazon、Facebook、Twitter/Digits、Google 或兼容 OpenID Connect 的任何提供商)，以及未经身份验证的身份。Amazon Cognito 还支持[已经过开发人员验证的身份](#)，借助该身份，您可以注册用户并通过自己的后端身份验证流程对用户进行身份验证，同时仍然使用 Amazon Cognito Sync 同步用户数据和访问 AWS 资源。

有关 [Amazon Cognito](#) 的更多信息，请参阅 [Amazon Cognito 开发人员指南](#)

以下代码示例显示如何轻松使用 Amazon Cognito 身份。[凭证提供程序](#) 示例显示如何创建用户身份并对其进行身份验证。[CognitoAuthentication 扩展库](#) 示例显示如何使用 CognitoAuthentication 扩展库来对 Amazon Cognito 用户群体进行身份验证。

主题

- [Amazon Cognito 凭证提供程序](#)
- [Amazon CognitoAuthentication 扩展库示例](#)

Amazon Cognito 凭证提供程序

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

`Amazon.CognitoIdentity.CognitoAWSCredentials`，位于 [AWSSDK.CognitoIdentity](#) NuGet 程序包中，是一个凭证对象，它使用 Amazon Cognito 和 AWS Security Token Service (AWS STS) 来检索用于进行 AWS 调用的凭证。

设置 `CognitoAWSCredentials` 的第一步是创建一个“身份池”。(身份池是用于存储特定于您的账户的用户身份信息的存储区。此信息可跨各种客户端平台、设备和操作系统进行检索，因此，如果用户在手机上开始使用此应用程序不久后又切换到平板电脑，该用户仍然可以使用保留的应用程序信息。您可以通过 Amazon Cognito 控制台创建新的身份池。如果您使用的是此控制台，它还将为您提供您所需的其他信息：

- 您的账号 - 一个 12 位数字，如 123456789012，这是您账户独有的号码。
- 未经身份验证的角色 ARN - 未经身份验证的用户将担任的角色。例如，此角色可提供对您的数据的只读权限。

- 经过身份验证的角色 ARN- 经过身份验证的用户将担任的角色。此角色可提供对您的数据的更广泛的权限。

设置 CognitoAWSCredentials

以下代码示例显示如何设置 `CognitoAWSCredentials`，然后您可使用此凭证以未经身份验证的角色身份调用 Amazon S3。这使您能够只需要对用户进行身份验证所需的最少量数据即可进行调用。用户权限由角色控制，因此您可以根据需要配置访问权限。

```
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();
}
```

以未经身份验证的用户身份使用 AWS

以下代码示例显示如何以未经身份验证的用户身份开始使用 AWS，然后通过 Facebook 进行身份验证并将凭证更新为使用 Facebook 凭证。使用此方法，您可以通过经过身份验证的角色授予经过身份验证的用户不同的功能。例如，您可能有一个电话应用程序，该应用程序允许用户匿名查看内容，但在用户使用一个或多个已配置的提供商登录的情况下允许其发帖。

```
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();
}
```

如果您将 `CognitoAWSCredentials` 对象与属于 AWS SDK for .NET 的一部分的 `AmazonCognitoSyncClient` 结合使用，则该对象会提供更多功能。如果您使用的是 `AmazonCognitoSyncClient` 和 `CognitoAWSCredentials`，则无需在使用 `IdentityPoolId` 进行调用时指定 `IdentityId` 和 `AmazonCognitoSyncClient` 属性。这些属性会自动从 `CognitoAWSCredentials` 中进行填入。以下代码示例将对此进行说明以及说明每当 `IdentityId` 的 `CognitoAWSCredentials` 发生更改时通知您的事件。在某些情况下 `IdentityId` 可能会发生更改，例如，当从未经身份验证的用户更改为经过身份验证的用户时。

```
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 = "..."
    });
}
```

Amazon CognitoAuthentication 扩展库示例

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

CognitoAuthentication 扩展库，位于[Amazon.Extensions.CognitoAuthentication](#) NuGet 程序包中，简化了 .NET Core 和 Xamarin 开发人员的 Amazon Cognito 用户群体身份验证过程。该库基于 Amazon Cognito 身份提供商 API 构建，可创建和发送用户身份验证 API 调用。

使用 CognitoAuthentication 扩展库

Amazon Cognito 有一些适用于标准身份验证流程的内置 AuthFlow 和 ChallengeName 值，以通过安全远程密码 (SRP) 协议验证用户名和密码。有关身份验证流程的更多信息，请参阅[Amazon Cognito 用户池身份验证流程](#)。

以下示例需要这些 using 语句：

```
// 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;
```

用户基本身份验证

使用不需要已签名请求的[AnonymousAWSCredentials](#) 创建一个[AmazonCognitoIdentityProviderClient](#)。您无需提供一个区域，如果未提供区域，底层代码将调用 `FallbackRegionFactory.GetRegionEndpoint()`。创建 `CognitoUserPool` 和 `CognitoUser` 对象。使用包含用户密码的 `StartWithSrpAuthAsync` 调用 `InitiateSrpAuthRequest` 方法。

```
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;
}
```

使用质询进行身份验证

使用质询继续进行身份验证流程 (如使用 `NewPasswordRequired` 和多重验证 (MFA)) 也较为简单。唯一需要的是 `CognitoAuthentication` 对象、用户的 SRP 密码以及进行下一个质询所需的信息，这是在提示用户进行输入时获取的。以下代码显示了一种在身份验证流程期间检查质询类型并获取针对 MFA 和 `NewPasswordRequired` 质询的相应响应的方法。

像之前一样执行基本身份验证请求，并且 `await` 一个 `AuthFlowResponse`。当收到响应时，循环访问返回的 `AuthenticationResult` 对象。如果 `ChallengeName` 类型为 `NEW_PASSWORD_REQUIRED`，请调用 `RespondToNewPasswordRequiredAsync` 方法。

```
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.");
}
```

```
}
```

在进行身份验证后使用 AWS 资源

在使用 `CognitoAuthentication` 库对某个用户进行身份验证后，下一步是允许该用户访问适当的 AWS 资源。为此，您必须通过 Amazon Cognito 联合身份控制台创建一个身份池。通过将您创建的 Amazon Cognito 用户池指定为提供商并使用其 `poolID` 和 `clientID`，您可以允许您的 Amazon Cognito 用户群体用户访问连接到您账户的 AWS 资源。您还可以指定不同的角色来支持未经身份验证的和已经过身份验证的用户访问不同的资源。您可以在 IAM 控制台中更改这些规则，其中，您可以在角色的附加策略的操作字段中添加或删除权限。然后，使用适当的身份池、用户群体和 Amazon Cognito 用户信息，您可以调用不同的 AWS 资源。以下示例显示了使用用于访问关联的身份池的角色允许的不同 Amazon S3 桶的 SRP 进行用户身份验证

```
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);
        }
    }
}
```

```
}  
}
```

更多身份验证选项

除了 SRP、NewPasswordRequired 和 MFA 以外，CognitoAuthentication 扩展库还提供了针对以下内容的更简单的身份验证流程：

- 自定义- 通过调用 `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)` 启动
- RefreshToken - 通过调用 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` 启动
- RefreshTokenSRP - 通过调用 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` 启动
- AdminNoSRP - 通过调用 `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)` 启动

根据您想要的流程调用相应的方法。然后，当质询显示在每个方法调用的 `AuthFlowResponse` 对象中时继续提示用户进行质询。此外，调用相应的响应方法，如适用于 MFA 质询的 `RespondToSmsMfaAuthAsync` 和适用于自定义质询的 `RespondToCustomAuthAsync`。

使用 Amazon DynamoDB NoSQL 数据库

Note

这些主题中的编程模型同时存在于 .NET Framework 和 .NET (Core) 中，但是调用约定不同，无论是同步还是异步。

AWS SDK for .NET 支持 Amazon DynamoDB，这是一项由 AWS 提供的快速 NoSQL 数据库服务。开发工具包为与 DynamoDB 通信提供了三种编程模型：低级模型、文档模型和对象持久性模型。

以下信息介绍了这些模型及其 API，提供有关使用方式和使用时的示例，并向您提供 AWS SDK for .NET 中额外 DynamoDB 编程资源的链接。

主题

- [低级模型](#)
- [文档模型](#)
- [对象持久化模型](#)
- [更多信息](#)
- [将表达式用于 Amazon DynamoDB 以及AWS SDK for .NET](#)
- [Amazon DynamoDB 中的 JSON 支持](#)

低级模型

低级编程模型封装对 DynamoDB 服务的直接调用。您可以通过 [Amazon.DynamoDBv2](#) 命名空间访问此模型。

在三种模型中，低级模型需要编写的代码最多。例如，您必须将 .NET 数据类型转换为 DynamoDB 中的对等类型。但是，此模型向您提供了对大部分功能的访问。

以下示例显示如何使用低级模型创建表、修改表以及将项目插入到 DynamoDB 中的表。

创建表

在以下示例中，您可以使用 CreateTable 类的 AmazonDynamoDBClient 方法创建表。CreateTable 方法使用包含特性的 CreateTableRequest 类的实例，例如必需的项目属性名称、主键定义和吞吐容量。该 CreateTable 方法返回 CreateTableResponse 类的实例。

```
// 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);
}
```

验证该表已准备好修改

您必须先准备好表进行修改，然后才能更改或修改表。以下示例显示了如何使用低级模型验证 DynamoDB 中的表已准备就绪。在此示例中，通过 DescribeTable 类的

AmazonDynamoDBClient 方法引用要检查的目标表。该代码每五秒检查一次表的 TableStatus 属性的值。当状态设置为 ACTIVE 时，表已准备好供修改。

```
// 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);
```

将项目插入到表中

在以下示例中，您使用低级模型将两个项目插入到 DynamoDB 中的表。每个项目通过 PutItem 类的 AmazonDynamoDBClient 方法插入，使用 PutItemRequest 类的实例。PutItemRequest 类的两个实例中的每个实例都接受将要插入项目的表名以及一系列项目属性值。

```
// 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);
```

文档模型

文档编程模型提供了一种更简单的方法来处理 DynamoDB 中的数据。此模型专门用于访问表和表中的项目。您可以通过 [Amazon.DynamoDBv2.DocumentModel](#) 命名空间访问此模型。

与低级编程模型相比，使用文档模型更容易针对 DynamoDB 数据编写代码。例如，您无需将相同数量的 .NET 数据类型转换为 DynamoDB 中的对等类型。不过，使用此模型所能访问的功能没有低级编程模型所提供的多。例如，您可以使用此模型创建、检索、更新和删除表中的项目。但是，要创建表，您必须使用低级模型。与对象持久化模型相比，此模型需要编写更多代码来存储、加载和查询 .NET 对象。

有关 DynamoDB 文档编程模型的更多信息，请参阅 [Amazon DynamoDB 开发人员指南](#) 中的 [.NET : 文档模型](#)。

以下各节提供有关如何创建所需的 DynamoDB 表的表示形式的信息，以及如何使用文档模型将项目插入表和从表中获取项目的示例。

创建表的表示形式

要使用文档模型执行数据操作，您必须首先创建 `Table` 类的实例，它会代表特定的表。有两种主要方式可执行此操作：

LoadTable 方法

第一种机制是使用 `Table` 类的静态 `LoadTable` 方法之一，类似于以下示例：

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

Note

虽然这种机制起作用，但在某些条件下，由于冷启动和线程池行为，它有时会导致额外的延迟或死锁。有关这些行为的更多信息，请参阅博客文章 [Improved DynamoDB Initialization Patterns for the AWS SDK for .NET](#)。

TableBuilder

在 [AWSSDK.DynamoDBv2 NuGet 程序包的 3.7.203 版本](#) 中引入了另一种机制，即 `TableBuilder` 类。该机制可以通过删除某些隐式方法调用（特别是 `DescribeTable` 方法）来解决上述行为。此机制的使用方式类似于以下示例：

```
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();
```

有关此替代机制的更多信息，请再次参阅博客文章 [Improved DynamoDB Initialization Patterns for the AWS SDK for .NET](#)。

将项目插入到表中

在以下示例中，回复通过 `Table` 类的 `PutItemAsync` 方法插入到 `Reply` 表中。`PutItemAsync` 方法获取 `Document` 类的实例；`Document` 类只是已初始化属性的集合。

```
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);
```

从表中获取项目

在以下示例中，通过 `Table` 类的 `GetItemAsync` 方法检索回复。为确定要获取的回复，`GetItemAsync` 方法使用目标回复的 `hash-and-range` 主键。

```
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"]);
```

前面的示例为 `WriteLine` 方法隐式将表值转换为字符串。您可以使用 `DynamoDBEntry` 类的多种 `As[type]` 方法执行显式转换。例如，对于 `Id` 的值，您可以通过 `AsGuid()` 方法将其从 `Primitive` 数据类型显式转换为 `GUID`：

```
var guid = reply["Id"].AsGuid();
```

对象持久化模型

对象持久性编程模型专门针对在 DynamoDB 中存储、加载和查询 .NET 对象而设计。您可以通过 [Amazon.DynamoDBv2.DataModel](#) 命名空间访问此模型。

在三种模型中，使用对象持久性模型最容易针对要存储、加载或查询的 DynamoDB 数据编写代码。例如，您可以直接使用 DynamoDB 数据类型。但是，此模型仅支持在 DynamoDB 中存储、加载和查询 .NET 对象的操作。例如，您可以使用此模型创建、检索、更新和删除表中的项目。不过，您必须先使用低级模型创建表，然后使用此模型将 .NET 类映射到表。

有关 DynamoDB 对象持久性编程模型的更多信息，请参阅 [Amazon DynamoDB 开发人员指南](#) 中的 [.NET : 对象持久性模型](#)。

以下示例向您演示如何定义表示 DynamoDB 项目的 .NET 类，使用 .NET 类的实例将项目插入到 DynamoDB ，以及使用 .NET 对象的实例从表获取项目。

定义表示表中项目的 .NET 类

在以下类定义示例中，DynamoDBTable 属性指定表名，而 DynamoDBHashKey 和 DynamoDBRangeKey 属性对表的 hash-and-range 主键进行建模。定义 DynamoDBGlobalSecondaryIndexHashKey 属性的目的是为了可以构造对特定作者回复的查询。

```
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; }
}
```

```
}
```

为对象持久性模型创建上下文

要使用适用于 DynamoDB 的对象持久性编程模型，您需要创建一个上下文，它提供到 DynamoDB 的连接，让您能够访问表、执行各种操作，以及执行查询。

基本上下文

以下代码示例演示如何创建最基本的上下文。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

具有 DisableFetchingTableMetadata 属性的上下文

以下示例显示了如何额外设置 DynamoDBContextConfig 类的 DisableFetchingTableMetadata 属性以防止隐式调用 DescribeTable 方法。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

如果将该 DisableFetchingTableMetadata 属性设置为 false (默认值)，如第一个示例所示，则可以省略 Reply 类中描述表项键和索引结构的属性。相反，这些属性将通过 DescribeTable 方法的隐式调用推断出来。如第二个示例所示，如果 DisableFetchingTableMetadata 设置为 true，则对象持久性模型的方法 (例如 SaveAsync 和 QueryAsync) 完全依赖于 Reply 类中定义的属性。在这种情况下，不会调用 DescribeTable 方法。

Note

在某些情况下，由于冷启动和线程池行为，对 DescribeTable 方法的调用有时会导致额外的延迟或死锁。因此，有时避免调用该方法是有利的。

有关这些行为的更多信息，请参阅博客文章 [Improved DynamoDB Initialization Patterns for the AWS SDK for .NET](#)。

使用 .NET 类的实例将项目插入到表中

在本示例中，项目通过 SaveAsync 类的 DynamoDBContext 方法插入，该方法采用表示项目的 .NET 类的已初始化实例。

```
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"
});
```

使用 .NET 类的实例从表中获取项目

在此示例中，使用 DynamoDBContext 类的 QueryAsync 方法创建了一个查询，用于查找“Author1”的所有记录。然后，通过查询的 GetNextSetAsync 方法检索项目。

```
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);
}
```

有关对象持久性模型的其他信息

上面显示的示例和解释有时包括名为 `DisableFetchingTableMetadata` 的 `DynamoDBContext` 类的属性。此属性在 [AWSSDK.DynamoDBv2 NuGet 程序包的 3.7.203 版本](#) 中引入，允许您避免某些可能由于冷启动和线程池行为而导致额外延迟或死锁的情况。有关更多信息，请参阅博客文章 [Improved DynamoDB Initialization Patterns for the AWS SDK for .NET](#)。

以下是有关此属性的一些其它信息。

- 如果您使用 .NET Framework，则可以在 `app.config` 或 `web.config` 文件中全局设置此属性。
- 可以使用 [AWSConfigsDynamoDB](#) 类全局设置此属性，如下面的示例所示。

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- 在某些情况下，您无法将 `DynamoDB` 属性添加到 .NET 类中；例如，如果该类是在依赖项中定义的。在这种情况下，仍然可以利用 `DisableFetchingTableMetadata` 属性。为此，除了 `DisableFetchingTableMetadata` 属性之外，还要使用 [TableBuilder](#) 类。在 [AWSSDK.DynamoDBv2 NuGet 程序包的 3.7.203 版本](#) 中引入了 `TableBuilder` 类。

```
// 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"
});
```

更多信息

使用AWS SDK for .NET编程 DynamoDB 信息和示例**

- [DynamoDB API](#)
- [DynamoDB 系列入门](#)
- [DynamoDB 系列 - 文档模型](#)
- [DynamoDB 系列 - 转换架构](#)
- [DynamoDB 系列 - 对象持久化模型](#)
- [DynamoDB 系列 - 表达式](#)
- [将表达式用于 Amazon DynamoDB 以及AWS SDK for .NET](#)
- [Amazon DynamoDB 中的 JSON 支持](#)

低级模型信息和示例

- [使用AWS SDK for .NET低级 API 处理表](#)

- [使用AWS SDK for .NET低级 API 处理项目](#)
- [使用AWS SDK for .NET低级 API 查询表](#)
- [使用AWS SDK for .NET低级 API 扫描表](#)
- [示例：使用AWS SDK for .NET低级 API 处理本地二级索引](#)
- [使用AWS SDK for .NET低级 API 处理全局二级索引](#)

文档模型信息和示例

- [DynamoDB 数据类型](#)
- [DynamoDBEntry](#)
- [.NET：文档模型](#)

对象持久性模型信息和示例

- [.NET：对象持久化模型](#)

将表达式用于 Amazon DynamoDB 以及AWS SDK for .NET

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

以下代码示例演示如何使用AWS SDK for .NET通过表达式对 DynamoDB 进行编程。表达式表示您希望从 DynamoDB 表中的某个项目读取到的属性。您还可以在编写项目时使用表达式指示任意必需满足的条件（也称为有条件更新），以及需要如何更新属性。一些更新示例使用新值替换属性，或者将新数据添加到列表或映射。有关更多信息，请参阅[使用表达式读取和写入项目](#)。

主题

- [示例数据](#)
- [使用表达式和项目的主键获取单个项目](#)
- [使用表达式和表主键获取多个项目](#)
- [使用表达式和其他项目属性获取多个项目](#)
- [输出项目](#)

- [使用表达式创建或替换项目](#)
- [使用表达式更新项目](#)
- [使用表达式删除项目](#)
- [更多信息](#)

示例数据

此主题中的代码示例依赖于名为 ProductCatalog 的 DynamoDB 表中的以下两个示例项目。这些项目描述一家虚构自行车商店目录中产品条目的相关信息。这些项目以[案例研究：ProductCatalog 项目](#)中提供的示例为基础。BOOL、L、M、N、NS、S 和 SS 等数据类型描述符对应于 [JSON 数据格式](#)中的那些描述符。

```
{
  "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."
      ]
    }
  }
}
```

使用表达式和项目的主键获取单个项目

以下示例介绍了 `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` 方法和一组表达式，用于获取并输出 Id 为 205 的项目。只返回项目的以下属性：`Id`、`Title`、`Description`、`Color`、`RelatedItems`、`Pictures` 和 `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);
```

在上述示例中，`ProjectionExpression` 属性指定要返回的属性。`ExpressionAttributeNames` 属性指定占位符 `#pr` (表示 `ProductReviews` 属性) 和占位符 `#ri` (表示 `RelatedItems` 属性)。对 `PrintItem` 的调用引用自定义函数，如[输出项目](#)中所述。

使用表达式和表主键获取多个项目

以下示例介绍了 `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` 方法和一组表达式，仅在 `Id` 的值大于 301 时，获取并输出 `Price` 为 150 的项目。只返回项目的以下属性：`Id`、`Title` 以及所有 `ThreeStar` 中的所有 `ProductReviews` 属性。

```
// 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("====");
}
```

在上述示例中，`ProjectionExpression` 属性指定要返回的属性。`ExpressionAttributeNames` 属性指定占位符 `#pr`（表示 `ProductReviews` 属性）和占位符 `#p`（表示 `Price` 属性）。`#pr.ThreeStar` 指定只返回 `ThreeStar` 属性。`ExpressionAttributeValues` 属性指定占位符 `:val` 来表示值 `150`。`FilterExpression` 属性指定 `#p (Price)` 必须大于 `:val (150)`。对 `PrintItem` 的调用引用自定义函数，如[输出项目](#)中所述。

使用表达式和其他项目属性获取多个项目

以下示例介绍了 `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` 方法和一组表达式，用于获取并输出 `ProductCategory` 为 `Bike` 的所有项目。只返回项目的以下属性：`Id`、`Title` 以及 `ProductReviews` 中的所有属性。

```
// 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("====");
}
```

在上述示例中，`ProjectionExpression` 属性指定要返回的属性。`ExpressionAttributeNames` 属性指定占位符 `#pr` (表示 `ProductReviews` 属性) 和占位符 `#pc` (表示 `ProductCategory` 属性)。`ExpressionAttributeValues` 属性指定占位符 `:catg` 来表示值 `Bike`。`FilterExpression` 属性指定 `#pc` (`ProductCategory`) 必须等于 `:catg` (`Bike`)。对 `PrintItem` 的调用引用自定义函数，如[输出项目](#)中所述。

输出项目

以下示例演示如何输出项目的属性和值。此示例已在前面的几个示例中使用过，这些示例演示如何[使用表达式和项目的主键获取单个项目](#)、[使用表达式和表主键获取多个项目](#)以及[使用表达式和其他项目属性获取多个项目](#)。

```
// 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.WriteLine("\n");
    PrintItem(value.M);
}
// Number attribute value.
else if (value.N != null)
{
    Console.WriteLine(value.N);
}
// Number set attribute value.
else if (value.NS.Count > 0)
{
    Console.WriteLine("{0}", string.Join("\n", value.NS.ToArray()));
}
// Null attribute value.
else if (value.NULL)
{
    Console.WriteLine("Null");
}
// String attribute value.
else if (value.S != null)
{
    Console.WriteLine(value.S);
}
// String set attribute value.
else if (value.SS.Count > 0)
{
    Console.WriteLine("{0}", string.Join("\n", value.SS.ToArray()));
}
// Otherwise, boolean value.
else
{
    Console.WriteLine(value.BOOL);
}
```

```
}

    Console.WriteLine("\n");
}
```

在上述示例中，每个属性值具有多个数据/类型特定的属性，可以进行求值来确定要用于输出属性的正确格式。这些属性包括 B、BOOL、BS、L、M、N、NS、NULL、S 和 SS，对应于 [JSON 数据格式](#) 中的那些属性。对于 B、N、NULL 和 S 等属性，如果对应属性不是 null，则属性是对应的非 null 数据类型。对于 BS、L、M、NS 和 SS 等属性，如果 Count 大于零，则属性是对应的非零值数据类型。如果所有属性的数据/类型特定属性为 null，或 Count 等于零，则属性对应于 BOOL 数据类型。

使用表达式创建或替换项目

以下示例介绍了 Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem 方法和一组表达式，用于更新 Title 为 18-Bicycle 301 的项目。如果项目不存在，将添加新项目。

```
// 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);
```

在上述示例中，ExpressionAttributeNames 属性指定占位符 #title，以表示 Title 属性。ExpressionAttributeValues 属性指定占位符 :product 来表示值 18-Bicycle 301。ConditionExpression 属性指定 #title (Title) 必须等于 :product (18-Bicycle 301)。对 CreateItemData 的调用引用以下自定义函数：

```

// 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;
}

```

在上述示例中，具有示例数据的示例项目返回到调用方。构建了一系列属性和对应值，使用 BOOL、L、M、N、NS、S 和 SS 等数据类型，它们对应于 [JSON 数据格式](#) 中的那些类型。

使用表达式更新项目

以下示例介绍了 `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` 方法和一组表达式，用于将 Title 为 18" Girl's Bike 的项目的 Id 更改为 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);
```

在上述示例中，`ExpressionAttributeNames` 属性指定占位符 `#title`，以表示 `Title` 属性。`ExpressionAttributeValues` 属性指定占位符 `:newproduct` 来表示值 18" Girl's Bike。`UpdateExpression` 属性指定将 `#title` (`Title`) 更改为 `:newproduct` (18" Girl's Bike)。

使用表达式删除项目

以下示例介绍了 `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` 方法和一组表达式，仅在项目的 Title 为 18-Bicycle 301 时，删除 Id 为 301 的项目。

```
// 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);
```

在上述示例中，`ExpressionAttributeNames` 属性指定占位符 `#title`，以表示 `Title` 属性。`ExpressionAttributeValues` 属性指定占位符 `:product` 来表示值 `18-Bicycle 301`。`ConditionExpression` 属性指定 `#title (Title)` 必须等于 `:product (18-Bicycle 301)`。

更多信息

有关更多信息以及代码示例，请参阅：

- [DynamoDB 系列 - 表达式](#)
- [使用投影表达式访问项目属性](#)
- [为属性名称和值使用占位符](#)
- [使用条件表达式指定条件](#)
- [使用更新表达式修改项目和属性](#)
- [使用AWS SDK for .NET低级 API 处理项目](#)
- [使用AWS SDK for .NET低级 API 查询表](#)
- [使用AWS SDK for .NET低级 API 扫描表](#)

- [示例：使用AWS SDK for .NET低级 API 处理本地二级索引](#)
- [使用AWS SDK for .NET低级 API 处理全局二级索引](#)

Amazon DynamoDB 中的 JSON 支持

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

在使用 Amazon DynamoDB 时，AWS SDK for .NET支持 JSON 数据。这使您能够更轻松地从 DynamoDB 中获取 JSON 格式的数据以及将 JSON 文档插入到其中。

主题

- [从 DynamoDB 表以 JSON 格式获取数据](#)
- [将 JSON 格式数据插入 DynamoDB 表](#)
- [DynamoDB 数据类型转换为 JSON](#)
- [更多信息](#)

从 DynamoDB 表以 JSON 格式获取数据

以下示例显示了如何以 JSON 格式从 DynamoDB 表中获取数据：

```
// 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.WriteLine(jsonText);

// Output:
// {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
```



```
Console.WriteLine(jsonPrettyText);

// Output:
// {
//   "Name" : "Shadow",
//   "Type" : "Horse",
//   "Id"   : 3
// }
```

在上述示例中，Document 类的 ToJson 方法将表中的项目转换为 JSON 格式的字符串。项目通过 Table 类的 GetItem 方法检索。在此示例中，为确定要获取的项目，GetItem 方法使用目标项目的 hash-and-range 主键。为确定要从中获取项目的表，Table 类的 LoadTable 方法使用 AmazonDynamoDBClient 类的实例以及 DynamoDB 中的目标表名。

将 JSON 格式数据插入 DynamoDB 表

以下示例演示如何使用 JSON 格式将项目插入到 DynamoDB 表：

```
// 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);
```

在上述示例中，FromJson 类的 Document 方法将 JSON 格式的字符串转换为项目。项目通过 PutItem 类的 Table 方法插入表中，该方法使用包含项目的 Document 类的实例。为确定要插入项目的表，调用 Table 类的 LoadTable 方法，并指定 AmazonDynamoDBClient 类的实例以及 DynamoDB 中的目标表名。

DynamoDB 数据类型转换为 JSON

当您调用 Document 类的 ToJson 方法，然后在生成的 JSON 数据上调用 FromJson 方法以将 JSON 数据转换回 Document 类的实例时，一些 DynamoDB 数据类型不会按预期转换。具体来说：

- DynamoDB 集 (SS、NS 和 BS 类型) 将转换为 JSON 数组。
- DynamoDB 二进制标量和集 (B 和 BS 类型) 将转换为 base64 编码的 JSON 字符串或字符串列表。

在此情况下，您必须调用 `DecodeBase64Attributes` 类的 `Document` 方法，使用正确的二进制表示形式替换 base64 编码的 JSON 数据。以下示例使用正确的二进制表示形式，在 `Document` 类的实例中，替换 base64 编码的名为 `Picture` 的二进制标量项目属性。此示例还在 `Document` 类的相同实例中，为名为 `RelatedPictures` 的 base64 编码二进制集项目属性执行相同的操作。

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

更多信息

有关使用 AWS SDK for .NET 通过 DynamoDB 进行 JSON 编程的更多信息和示例，请参阅：

- [DynamoDB JSON 支持](#)
- [Amazon DynamoDB 更新 - JSON、更广泛的免费套餐、灵活扩展、更大的项目](#)

使用 Amazon EC2

AWS SDK for .NET 支持 [Amazon EC2](#)，这是一项提供可调整计算容量的网络服务。您可以使用这种计算容量来构建和托管您的软件系统。

API

AWS SDK for .NET 为 Amazon EC2 客户提供了 API。这些 API 使您能够使用安全组和密钥对等 EC2 功能。API 还使您能够控制 Amazon EC2 实例。本节包含少量示例，向您展示使用这些 API 时可以遵循的模式。要查看完整的 API 集，请参阅 [AWS SDK for .NET API 参考](#)（并滚动至“Amazon.EC2”）。

Amazon EC2 API 由 [AWSSDK.EC2](#) NuGet 程序包提供。

先决条件

开始之前，请确保您已 [完成环境和项目的设置](#)。还要查看 [软件开发工具包功能](#) 中的信息。

关于示例

本节中的示例向您展示如何使用 Amazon EC2 客户端和管理 Amazon EC2 实例。

[EC2 竞价型实例教程](#) 向您演示如何请求 Amazon EC2 竞价型实例。竞价型实例允许您以低于按需价格访问未使用的 EC2 容量。

主题

- [使用 Amazon EC2 中的安全组](#)
- [使用 Amazon EC2 密钥对](#)
- [设置 Amazon EC2 区域和可用区](#)
- [使用 Amazon EC2 实例](#)
- [Amazon EC2 竞价型实例教程](#)

使用 Amazon EC2 中的安全组

在 Amazon EC2 中，安全组起着虚拟防火墙的作用，可控制一个或多个 EC2 实例的网络流量。默认情况下，EC2 将您的实例与不允许入站流量的安全组关联。可以创建允许您的 EC2 实例接受特定流量的安全组。例如，如果您需要连接到 EC2 Windows 实例，就必须配置安全组以允许 RDP 流量。

有关安全组的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)和[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

使用 AWS SDK for .NET 时，您可以创建在 EC2-VPC 或 EC2-Classical 中使用的安全组。有关 EC2-VPC 与 EC2-Classical 的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

Warning

我们将于 2022 年 8 月 15 日停用 EC2-Classical。我们建议您从 EC2-Classical 迁移到 VPC。有关更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中的从 EC2-Classical 迁移到 VPC。另请参阅 [EC2-Classical Networking is Retiring – Here's How to Prepare](#) 博客文章。

有关 API 和先决条件的信息，请参阅父部分 ([使用 Amazon EC2](#))。

主题

- [枚举安全组](#)
- [创建安全组](#)
- [更新安全组](#)

枚举安全组

此示例演示如何使用AWS SDK for .NET枚举安全组。如果您提供 [Amazon Virtual Private Cloud ID](#) ，则应用程序会枚举该特定 VPC 的安全组。否则，应用程序仅显示所有可用安全组的列表。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [枚举安全组](#)
- [完整代码](#)
- [其他注意事项](#)

枚举安全组

以下代码片段枚举了您的安全组。它枚举了特定 VPC (如果给出) 的所有组或组。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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($"
Getting security groups for VPC {vpcID}...
");
        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();
}
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)
 - 类 [AmazonEC2Client](#)
- 命名空间 [Amazon.EC2.Model](#)
 - 类 [DescribeSecurityGroupsRequest](#)
 - 类 [DescribeSecurityGroupsResponse](#)
 - 类 [Filter](#)
 - 类 [SecurityGroup](#)

代码

```
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();
}
}
}
}
```

其他注意事项

- 请注意 VPC 情况，筛选器是通过将名称值对的 Name 部分设置为“vpc-id”而构造的。这个名字来自 [DescribeSecurityGroupsRequest](#) 类 Filters 属性的描述。
- 可以使用 [无参数的 DescribeSecurityGroups](#) 获取安全组的完整列表。
- 您可以通过在 [Amazon EC2 控制台](#) 中查看安全组列表来验证结果。

创建安全组

此示例说明如何使用 AWS SDK for .NET 创建安全组。您可以提供现有 VPC 的 ID，以便在 VPC 中为 EC2 创建安全组。如果您不提供这样的 ID，则新的安全组将适用于 EC2-Classical (前提是您的 AWS 账户对此支持)。

如果您未提供 VPC ID 且您的 AWS 账户不支持 EC2-Classical，则新的安全组将属于您账户的默认 VPC。有关更多信息，请参阅父部分 ([使用 Amazon EC2 中的安全组](#)) 中有关 VPC 中的 EC2 与 EC2-Classical 的参考资料。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [查找现有安全组](#)
- [创建安全组](#)
- [完整代码](#)

查找现有安全组

以下代码片段搜索给定 VPC 中具有给定名称的现有安全组。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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;
}
```

创建安全组

如果给定的 VPC 中不存在具有该名称的组，则以下片段将创建一个新的安全组。如果未给出 VPC，并且存在一个或多个具有该名称的组，则该片段仅返回组列表。

[本主题末尾](#)的示例显示了此片段的使用情况。

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

    // If the security group doesn't already exist, create it.
    var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
    };
    if(string.IsNullOrEmpty(vpcID))
    {
        createRequest.Description = "My .NET example security group for EC2-Classic";
    }
    else
    {
        createRequest.VpcId = vpcID;
        createRequest.Description = "My .NET example security group for EC2-VPC";
    }
    CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

    // Return the new security group
    DescribeSecurityGroupsResponse describeResponse =
```

```
await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
    GroupIds = new List<string>() { createResponse.GroupId }
});
return describeResponse.SecurityGroups;
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)
 - 类 [AmazonEC2Client](#)
- 命名空间 [Amazon.EC2.Model](#)
 - 类 [CreateSecurityGroupRequest](#)
 - 类 [CreateSecurityGroupResponse](#)
 - 类 [DescribeSecurityGroupsRequest](#)
 - 类 [DescribeSecurityGroupsResponse](#)
 - 类 [Filter](#)
 - 类 [SecurityGroup](#)

代码

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2CreateSecGroup
{
    // = = = = =
    // Class to create a security group
    class Program
    {
        private const int MaxArgs = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }
            if(parsedArgs.Count > MaxArgs)
                CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Get the application arguments from the parsed list
            var groupName = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-name");
            var vpcID = CommandLine.GetArgument(parsedArgs, null, "-v", "--vpc-id");
            if(string.IsNullOrEmpty(groupName))
                CommandLine.ErrorExit("\nYou must supply a name for the new group." +
                    "\nRun the command with no arguments to see help.");
            if(!string.IsNullOrEmpty(vpcID) && !vpcID.StartsWith("vpc-"))
                CommandLine.ErrorExit($"Not a valid VPC ID: {vpcID}");

            // groupName has a value and vpcID either has a value or is null (which is fine)
            // Create the new security group and display information about it
            var securityGroups =
                await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
            Console.WriteLine("Information about the security group(s):");
            foreach(var group in securityGroups)
            {
                Console.WriteLine($"GroupName: {group.GroupName}");
                Console.WriteLine($"GroupId: {group.GroupId}");
                Console.WriteLine($"Description: {group.Description}");
                Console.WriteLine($"VpcId (if any): {group.VpcId}");
            }
        }
    }
}
```

```
}

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

    // If the security group doesn't already 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);
    }
}
}
```

更新安全组

此示例说明如何使用AWS SDK for .NET向安全组添加规则。特别是，该示例添加了一条规则，允许给定 TCP 端口上的入站流量，例如，该端口可用于到 EC2 实例的远程连接。应用程序获取现有安全组的 ID、CIDR 格式的 IP 地址（或地址范围）以及可选的 TCP 端口号。然后，它向给定的安全组添加入站规则。

Note

要使用此示例，您需要一个 CIDR 格式的 IP 地址（或地址范围）。有关获取本地计算机 IP 地址的方法，请参阅本主题末尾的其它注意事项。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [添加入站规则](#)
- [完整代码](#)
- [其他注意事项](#)

添加入站规则

以下片段向安全组添加了针对特定 IP 地址（或范围）和 TCP 端口的入站规则。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
    IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
    // Create an object to hold the request information for the rule.
    // It uses an IpPermission object to hold the IP information for the rule.
    var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
    ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
    });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"
New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}
```



```
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [AuthorizeSecurityGroupIngressRequest](#)

类 [AuthorizeSecurityGroupIngressResponse](#)

类 [IpPermission](#)

类 [IpRange](#)

代码

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2AddRuleForRDP
{
    // = = = = =
    // = = =
    // Class to add a rule that allows inbound traffic on TCP a port
    class Program
    {
```

```
private const int DefaultPort = 3389;

static async Task Main(string[] args)
{
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
        PrintHelp();
        return;
    }

    // Get the application arguments from the parsed list
    var groupID = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
    var ipAddress = CommandLine.GetArgument(parsedArgs, null, "-i", "--ip-address");
    var portStr = CommandLine.GetArgument(parsedArgs, DefaultPort.ToString(), "-p",
"--port");
    if(string.IsNullOrEmpty(ipAddress))
        CommandLine.ErrorExit("\nYou must supply an IP address in CIDR format.");
    if(string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
        CommandLine.ErrorExit("\nThe ID for a security group is missing or
incorrect.");
    if(int.Parse(portStr) == 0)
        CommandLine.ErrorExit($"The given TCP port number, {portStr}, isn't
allowed.");

    // Add a rule to the given security group that allows
    // inbound traffic on a TCP port
    await AddIngressRule(
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
}

//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
    IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
    // Create an object to hold the request information for the rule.
    // It uses an IpPermission object to hold the IP information for the rule.
    var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
    ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
```

```

        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
    });

    // Create the inbound rule for the security group
    AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
    Console.WriteLine($"New RDP rule was written in {groupID} for {ipAddress}.");
    Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}

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

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key

```

```
// (including the dashes).
// If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
// where "N" represents sequential numbers.
public static Dictionary<string,string> Parse(string[] args)
{
    var parsedArgs = new Dictionary<string,string>();
    int i = 0, n = 0;
    while(i < args.Length)
    {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
            var key = args[i++];
            var value = key;

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

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

    return parsedArgs;
}

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

```
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

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

其他注意事项

- 如果您不提供端口号，则应用程序默认为端口 3389。这是 Windows RDP 的端口，它使您能够连接到运行 Windows 的 EC2 实例。如果您要启动运行 Linux 的 EC2 实例，请改为使用 TCP 端口 22 (SSH)。
- 请注意，该示例将 `IpProtocol` 设置为“tcp”。`IpProtocol` 的值可以在 [IpPermission](#) 类的 `IpProtocol` 属性的描述中找到。
- 使用此示例时，您可能需要本地计算机的 IP 地址。以下是可以获取地址的一些方式。
 - 如果您的本地计算机（您将从该计算机连接到 EC2 实例）具有静态公有 IP 地址，则可以使用服务来获取该地址。其中一项服务是 <http://checkip.amazonaws.com/>。有关授权入栈流量的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。
 - 获取本地计算机 IP 地址的另一种方法是使用 [Amazon EC2 控制台](#)。

选择您的一个安全组，选择入站规则选项卡，然后选择编辑入站规则。在入站规则中，打开来源列表中的下拉菜单，然后选择我的 IP 以查看 CIDR 格式的本地计算机 IP 地址。请务必取消该操作。

- 您可以通过检查 [Amazon EC2 控制台](#) 中的安全组列表来验证此示例的结果。

使用 Amazon EC2 密钥对

Amazon EC2 使用公有密钥密码系统来加密和解密登录信息。公有密钥密码系统使用公有密钥加密数据，然后收件人可以使用私有密钥解密数据。公有和私有密钥被称为密钥对。如果要登录 EC2 实例，则必须在启动时指定密钥对，然后在连接时提供密钥对的私钥。

当您启动 EC2 实例时，您可以为其创建密钥对，或使用在启动其它实例时已使用过的密钥对。在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关 Amazon EC2 密钥对的更多信息。

有关 API 和先决条件的信息，请参阅父部分 ([使用 Amazon EC2](#))。

主题

- [创建和显示密钥对](#)
- [删除密钥对](#)

创建和显示密钥对

此示例演示如何使用 AWS SDK for .NET 创建密钥对。应用程序使用新密钥对的名称和 PEM 文件的名称 (扩展名为“.pem”)。其会创建密钥对，将私钥写入 PEM 文件，然后显示所有可用的密钥对。如果您不提供命令行参数，则应用程序仅显示所有可用的密钥对。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [创建密钥对](#)
- [显示可用的密钥对](#)
- [完整代码](#)
- [其他注意事项](#)

创建密钥对

以下代码片段创建了一个密钥对，然后将私钥存储到给定的 PEM 文件中。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// Method to create a key pair and save the key material in a PEM file
```

```
private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
    // Create the key pair
    CreateKeyPairResponse response =
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
            KeyName = keyPairName
        });
    Console.WriteLine($"Created new key pair: {response.KeyPair.KeyName}");

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

显示可用的密钥对

以下代码片段显示了可用密钥对的列表。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [CreateKeyPairRequest](#)

类 [CreateKeyPairResponse](#)

类 [DescribeKeyPairsResponse](#)

类 [KeyPairInfo](#)

代码

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
    // = = = = =
    // Class to create and store a key pair
    class Program
    {
        static async Task Main(string[] args)
        {
            // Create the EC2 client
            var ec2Client = new AmazonEC2Client();

            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                // In the case of no command-line arguments,
                // just show help and the existing key pairs
            }
        }
    }
}
```



```
PrintHelp();
Console.WriteLine("\nNo arguments specified.");
Console.Write(
    "Do you want to see a list of the existing key pairs? ((y) or n): ");
string response = Console.ReadLine();
if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
    await EnumerateKeyPairs(ec2Client);
return;
}

// Get the application arguments from the parsed list
string keyPairName =
    CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
string pemFileName =
    CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
if(string.IsNullOrEmpty(keyPairName))
    CommandLine.ErrorExit("\nNo key pair name specified." +
        "\nRun the command with no arguments to see help.");
if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
    CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

// Create the key pair
await CreateKeyPair(ec2Client, keyPairName, pemFileName);
await EnumerateKeyPairs(ec2Client);
}

//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
    // Create the key pair
    CreateKeyPairResponse response =
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
            KeyName = keyPairName
        });
    Console.WriteLine($"Created new key pair: {response.KeyPair.KeyName}");

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

```

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

//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n -k, --keypair-name: The name you want to assign to the key pair." +
        "\n -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
}
}

// = = = = =
// Class that represents a command line on the console or terminal.
// (This is the same for all examples. When you have seen it once, you can ignore
it.)
static class CommandLine
{
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
}

```

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

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

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

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
```

```
Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

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

其他注意事项

- 运行该示例后，您可以在 [Amazon EC2 控制台](#) 中看到新的密钥对。
- 创建密钥对时，必须保存返回的私钥，因为以后无法检索私钥。

删除密钥对

此示例说明如何使用AWS SDK for .NET删除密钥对。应用程序获取一个密钥对的名称。它会删除密钥对，然后显示所有可用的密钥对。如果您不提供命令行参数，则应用程序仅显示所有可用的密钥对。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [删除密钥对](#)
- [显示可用的密钥对](#)
- [完整代码](#)

删除密钥对

以下代码片段删除了密钥对。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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).");  
}
```

显示可用的密钥对

以下代码片段显示了可用密钥对的列表。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// Method to show the key pairs that are available  
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)  
{  
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();  
    Console.WriteLine("Available key pairs:");  
    foreach (KeyValuePair item in response.KeyPairs)  
        Console.WriteLine($" {item.KeyName}");  
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [DeleteKeyPairRequest](#)

类 [DescribeKeyPairsResponse](#)

类 [KeyPairInfo](#)

代码

```
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 (KeyValuePair item in response.KeyPairs)
        Console.WriteLine($" {item.KeyName}");
}
}
}
```

设置 Amazon EC2 区域和可用区

Amazon EC2 托管在全球多个位置。这些位置由 区域和可用区构成。每个区域都是一个独立的地理区域，具有多个相互隔离的位置，这些位置称为可用区。

在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关区域和可用区的更多信息。

此示例向您展示如何使用AWS SDK for .NET获取与 EC2 客户端相关的区域和可用区的详细信息。该应用程序显示 EC2 客户端可用的区域和可用区列表。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [DescribeAvailabilityZonesResponse](#)

类 [DescribeRegionsResponse](#)

类 [AvailabilityZone](#)

类 [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);
    }
}
}
```

使用 Amazon EC2 实例

您可以使用AWS SDK for .NET通过创建、启动和终止等操作来控制 Amazon EC2 实例。本节中的主题提供了一些如何执行此操作的示例。在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关 EC2 实例的更多信息。

有关 API 和先决条件的信息，请参阅父部分（[使用 Amazon EC2](#)）。

主题

- [启动 Amazon EC2 实例](#)
- [终止 Amazon EC2 实例](#)

启动 Amazon EC2 实例

此示例说明如何使用AWS SDK for .NET，从同一亚马逊机器映像（AMI）启动一个或多个配置相同的 Amazon EC2 实例。应用程序使用您提供的[多个输入](#)启动一个 EC2 实例，然后监控该实例，直到其退出“待处理”状态。

当您的 EC2 实例运行时，您可以远程连接到该实例，如 [\(可选 \) 连接到实例](#) 中所述。

您可以在 VPC 中启动 EC2，或如果您的AWS账户支持，可在 EC2-Classic 中启动。有关 EC2-VPC 与 EC2-Classic 的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

Warning

我们将于 2022 年 8 月 15 日停用 EC2-Classic。我们建议您从 EC2-Classic 迁移到 VPC。有关更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中的从 EC2-Classic 迁移到 VPC。另请参阅 [EC2-Classic Networking is Retiring – Here's How to Prepare](#) 博客文章。

以下各节提供了此示例的片段和其它信息。片段后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [收集所需内容](#)
- [启动 实例](#)
- [监控实例](#)
- [完整代码](#)
- [其他注意事项](#)
- [\(可选 \) 连接到实例](#)
- [清理](#)

收集所需内容

要启动 EC2 实例，您需要具备多项条件。

- 将在其中启动实例的 [VPC](#)。如果它是 Windows 实例，并且您将通过 RDP 连接到该实例，那么 VPC 很可能需要连接互联网网关，并在路由表中添加互联网网关的条目。有关更多信息，请参阅《Amazon VPC 用户指南》中的 [互联网网关](#)。
- 将在其中启动实例的 VPC 现有子网的 ID。要查找或创建，一种简单的方法是登录 [Amazon VPC 控制台](#)，不过也可以使用 [CreateSubnetAsync](#) 和 [DescribeSubnetsAsync](#) 方法以编程方式获取。

Note

如果您的 AWS 账户支持 EC2-Classic 并且这是您想要启动的实例类型，则不需要此参数。但是，如果您的账户不支持 EC2-Classic，并且您没有提供此参数，则新实例将在您账户的默认 VPC 中启动。

- 属于将在其中启动实例的 VPC 的现有安全组的 ID。有关更多信息，请参阅[使用 Amazon EC2 中的安全组](#)。
- 如果要连接到新实例，则前面提到的安全组必须有相应的入站规则，允许端口 22 (Linux 实例) 上的 SSH 流量或端口 3389 (Windows 实例) 上的 RDP 流量。有关如何执行此操作的信息，请参阅[更新安全组](#)，包括该主题接近末尾处的[其他注意事项](#)。
- 要用于创建实例的自定义亚马逊机器映像 (AMI) 的 ID。在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关 AMI 的更多信息。例如，在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关共享 AMI 的更多信息。
- 现有 EC2 密钥对的名称，用于连接到新实例。有关更多信息，请参阅[使用 Amazon EC2 密钥对](#)。
- 包含前面提到的 EC2 密钥对私钥的 PEM 文件的名称。当您[远程连接到](#)实例时，将使用 PEM 文件。

启动实例

以下代码片段启动 EC2 实例。

[本主题接近末尾处](#)的示例显示了此片段的使用情况。

```
//  
// 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;  
}
```

监控实例

以下代码片段会监控该实例，直到其退出“待处理”状态。

[本主题接近末尾处](#)的示例显示了此片段的使用情况。

有关 `Instance.State.Code` 属性的有效值，请参阅 [InstanceState](#) 类。

```
//  
// Method to wait until the instances are running (or at least not pending)  
private static async Task CheckState(IAmazonEC2 ec2Client, List<string>  
instanceIds)  
{  
    Console.WriteLine(  
        "\nWaiting for the instances to start." +  
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");  
  
    int numberRunning;  
    DescribeInstancesResponse responseDescribe;  
    var requestDescribe = new DescribeInstancesRequest{  
        InstanceIds = instanceIds};  
  
    // Check every couple of seconds
```

```
int wait = 2000;
while(true)
{
    // Get and check the status for each of the instances to see if it's past
    pending.
    // Once all instances are past pending, break out.
    // (For this example, we are assuming that there is only one reservation.)
    Console.WriteLine(".");
    numberRunning = 0;
    responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
    foreach(Instance i in responseDescribe.Reservations[0].Instances)
    {
        // Check the lower byte of State.Code property
        // Code == 0 is the pending state
        if((i.State.Code & 255) > 0) numberRunning++;
    }
    if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
        break;

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

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

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

类 [InstanceType](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [DescribeInstancesRequest](#)

类 [DescribeInstancesResponse](#)

类 [Instance](#)

类 [InstanceNetworkInterfaceSpecification](#)

类 [RunInstancesRequest](#)

类 [RunInstancesResponse](#)

代码

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
    // = = = = =
    // = = =
    // Class to launch an EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
```

```
var parsedArgs = CommandLine.Parse(args);
if(parsedArgs.Count == 0)
{
    PrintHelp();
    return;
}

// Get the application arguments from the parsed list
string groupID =
    CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
string ami =
    CommandLine.GetArgument(parsedArgs, null, "-a", "--ami-id");
string keyPairName =
    CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
string subnetID =
    CommandLine.GetArgument(parsedArgs, null, "-s", "--subnet-id");
if( (string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
    || (string.IsNullOrEmpty(ami) || !ami.StartsWith("ami-"))
    || (string.IsNullOrEmpty(keyPairName))
    || (!string.IsNullOrEmpty(subnetID) && !subnetID.StartsWith("subnet-")))
    CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

// Create an EC2 client
var ec2Client = new AmazonEC2Client();

// Create an object with the necessary properties
RunInstancesRequest request = GetRequestData(groupID, ami, keyPairName,
subnetID);

// Launch the instances and wait for them to start running
var instanceIds = await LaunchInstances(ec2Client, request);
await CheckState(ec2Client, instanceIds);
}

//
// Method to put together the properties needed to launch the instance.
private static RunInstancesRequest GetRequestData(
    string groupID, string ami, string keyPairName, string subnetID)
{
    // Common properties
    var groupIDs = new List<string>() { groupID };
}
```

```
var request = new RunInstancesRequest()
{
    // The first three of these would be additional command-line arguments or
similar.
    InstanceType = InstanceType.T1Micro,
    MinCount = 1,
    MaxCount = 1,
    ImageId = ami,
    KeyName = keyPairName
};

// Properties specifically for EC2 in a VPC.
if(!string.IsNullOrEmpty(subnetID))
{
    request.NetworkInterfaces =
        new List<InstanceNetworkInterfaceSpecification>() {
            new InstanceNetworkInterfaceSpecification() {
                DeviceIndex = 0,
                SubnetId = subnetID,
                Groups = groupIDs,
                AssociatePublicIpAddress = true
            }
        };
}

// Properties specifically for EC2-Classic
else
{
    request.SecurityGroupIds = groupIDs;
}
return request;
}

//
// Method to launch the instances
// Returns a list with the launched instance IDs
private static async Task<List<string>> LaunchInstances(
    IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
{
    var instanceIds = new List<string>();
    RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);
}
```



```
Console.WriteLine("\nNew instances have been created.");
foreach (Instance item in responseLaunch.Reservation.Instances)
{
    instanceIds.Add(item.InstanceId);
    Console.WriteLine($" New instance: {item.InstanceId}");
}

return instanceIds;
}

//
// Method to wait until the instances are running (or at least not pending)
private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
{
    Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

    int numberRunning;
    DescribeInstancesResponse responseDescribe;
    var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

    // Check every couple of seconds
    int wait = 2000;
    while(true)
    {
        // Get and check the status for each of the instances to see if it's past
        pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.WriteLine(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
            // Check the lower byte of State.Code property
            // Code == 0 is the pending state
            if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
            break;
    }
}
```

```

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

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

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
}
}

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

```

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

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

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

    return parsedArgs;
}

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


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

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

其他注意事项

- 检查 EC2 实例的状态时，您可以向 [DescribeInstancesRequest](#) 对象的 Filter 属性添加筛选条件。使用这种技术，您可以将请求限制为某些实例；例如，带有特定用户指定标签的实例。
- 为简洁起见，一些属性被赋予了典型值。这些属性中的任何或全部都可以通过编程方式或通过用户输入来确定。
- 您可以用于 [RunInstancesRequest](#) 对象的 MinCount 和 MaxCount 属性的值由目标可用区和该实例类型允许的最大实例数决定。有关更多信息，请参阅 Amazon EC2 一般常见问题解答中的 [我可以在 Amazon EC2 中运行多少个实例？](#)。
- 如果您想使用与本示例不同的实例类型，则有几类实例类型可供选择，您可以在 [适用于 Linux 实例的 Amazon EC2 用户指南](#) 或 [适用于 Windows 实例的 Amazon EC2 用户指南](#) 中查看这些类型。

- 您还可以在启动实例时将 [IAM 角色](#) 附加到实例。为此，请创建一个 [IamInstanceProfileSpecification](#) 对象，其 Name 属性设置为 IAM 角色的名称。然后将该对象添加到 [RunInstancesRequest](#) 对象的 `IamInstanceProfile` 属性中。

 Note

要启动附加了 IAM 角色的 EC2 实例，IAM 用户的配置必须包含某些权限。有关所需权限的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

(可选) 连接到实例

在实例运行之后，您可以使用合适的远程客户端远程连接该实例。对于 Linux 和 Windows 实例，您需要实例的公有 IP 地址或公有 DNS 名称。您还需要以下项目。

对于 Linux 实例

您可以使用 SSH 客户端连接到您的 Linux 实例。确保启动实例时使用的安全组允许端口 22 上的 SSH 流量，如[更新安全组](#)中所述。

您还需要用于启动实例的密钥对私有部分；即 PEM 文件。


有关更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的[连接到您的 Linux 实例](#)。

对于 Windows 实例

您可以使用 RDP 客户端连接到您的实例。确保启动实例时使用的安全组允许端口 3389 上的 RDP 流量，如[更新安全组](#)中所述。

您还需要管理员密码。您可以使用以下示例代码来获取此信息，该代码需要实例 ID 和用于启动实例的密钥对的私有部分，即 PEM 文件。

有关更多信息，请参阅《适用于 Windows 实例的 Amazon EC2 用户指南》中的[连接您的 Windows 实例](#)。

 Warning

此示例代码返回您实例的纯文本管理员密码。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)
 - 类 [AmazonEC2Client](#)
- 命名空间 [Amazon.EC2.Model](#)
 - 类 [GetPasswordDataRequest](#)
 - 类 [GetPasswordDataResponse](#)

代码

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
    // = = = = =
    // Class to get the Administrator password of a Windows EC2 instance
    class Program
    {
        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if(parsedArgs.Count == 0)
            {
                PrintHelp();
                return;
            }
        }
    }
}
```

```
// Get the application arguments from the parsed list
string instanceID =
    CommandLine.GetArgument(parsedArgs, null, "-i", "--instance-id");
string pemFileName =
    CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
if( (string.IsNullOrEmpty(instanceID) || !instanceID.StartsWith("i-"))
    || (string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem")))
    CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

// Create the EC2 client
var ec2Client = new AmazonEC2Client();

// Get and display the password
string password = await GetPassword(ec2Client, instanceID, pemFileName);
Console.WriteLine($"Password: {password}");
}

//
// Method to get the administrator password of a Windows EC2 instance
private static async Task<string> GetPassword(
    IAmazonEC2 ec2Client, string instanceID, string pemFilename)
{
    string password = string.Empty;
    GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
            InstanceId = instanceID});
    if(response.PasswordData != null)
    {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
    }
    else
    {
        Console.WriteLine($"The password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
    }
    return password;
}
```

```

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\n -i, --instance-id: The name of the EC2 instance." +
        "\n -p, --pem-filename: The name of the PEM file with the private key.");
}
}

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

```



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

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

return parsedArgs;
}

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

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

```
}
```

清理

您不再需要 EC2 实例时，请确保终止该实例，如[终止 Amazon EC2 实例](#)中所述。

终止 Amazon EC2 实例

如果您不再需要一个或多个 Amazon EC2 实例，则可终止这些实例。

此示例说明如何使用 AWS SDK for .NET 终止 EC2 实例。它以实例 ID 作为输入。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [TerminateInstancesRequest](#)

类 [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);
    }
}
}
```

运行该示例后，最好登录 [Amazon EC2 控制台](#) 来验证 [EC2 实例](#) 是否已终止。

Amazon EC2 竞价型实例教程

本教程介绍如何使用AWS SDK for .NET管理 Amazon EC2 竞价型实例。

概述

竞价型实例允许您以低于按需价格请求未使用的 Amazon EC2 容量。这可以显著降低可能被中断的应用程序的 EC2 成本。

下面简要概述了如何请求和使用竞价型实例。

1. 创建竞价型实例请求，指定您愿意支付的最高价格。
2. 请求完成后，请像运行任何其它 Amazon EC2 实例一样运行该实例。
3. 根据需要运行该实例，然后将其终止，除非 Spot 价格发生变化导致实例终止。
4. 在不再需要竞价型实例请求时对其进行清理，这样就不会再创建竞价型实例。

这是对竞价型实例整体层面的概述。您可以通过阅读[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)来更好地了解竞价型实例。

关于本教程

按本教程操作时，您可以使用 AWS SDK for .NET 执行以下操作：

- [创建竞价型实例请求](#)
- [确定何时执行该竞价型实例请求](#)
- [取消竞价型实例请求](#)
- [终止相关实例](#)

以下各节提供了此示例的片段和其它信息。片段后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [先决条件](#)
- [收集所需内容](#)
- [创建竞价型实例请求](#)
- [确定您的竞价型实例请求的状态](#)
- [清理您的竞价型实例请求](#)
- [清理您的竞价型实例](#)
- [完整代码](#)
- [其他注意事项](#)

先决条件

有关 API 和先决条件的信息，请参阅父部分 ([使用 Amazon EC2](#))。

收集所需内容

要创建竞价型实例请求，您需要一些东西。

- 实例数量和其实例类型。有多种实例类型可供选择，您可以在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中查看这些类型。在本教程中，默认数量为 1。
- 要用于创建实例的自定义亚马逊机器映像 (AMI) 的 ID。在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关 AMI 的更多信息。例如，在[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中了解有关共享 AMI 的更多信息。
- 您每小时愿意为每个实例支付的最高价格。您可以在 [Amazon EC2 定价页面](#)上查看所有实例类型 (包括按需实例和竞价型实例) 的价格。本教程的默认价格将在后面说明。
- 如果您想远程连接到实例，则需要具有适当配置和资源的安全组。[使用 Amazon EC2 中的安全组](#)中对此进行了描述，并在[启动 Amazon EC2 实例](#)中提供了有关[收集所需内容](#)和[连接到实例](#)的信息。为简单起见，本教程使用了所有新 AWS 账户都拥有的名为 default 的安全组。

有很多方法可以请求 Spot 实例。以下是常见策略：

- 提出请求以确保成本低于按需定价。
- 基于最终的计算值提出请求。
- 提出请求以便尽快获得计算容量。

关于以下说明，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中的 Spot 价格历史记录。

降低成本至低于按需价格

您需要进行花费数小时或数天的批处理工作。然而，您可以灵活调整启动和结束时间。您希望看到是否以低于按需实例的成本完成。

您可以通过使用 Amazon EC2 控制台或 Amazon EC2 API 来检查各个类型实例的 Spot 价格历史记录。在您分析了给定可用区内所需实例类型的价格记录之后，您有两种可供选择的方法发出请求：

- 指定在 Spot 价格范围（这仍然低于按需定价）的上限发出请求，预测您单次竞价型实例请求很有可能会达成，并运行足够的连续计算时间来完成此项工作。
- 指定在 Spot 价格范围的下限发出请求，随着时间的推移，通过持久的请求，计划结合多种已启动实例。总计一下，该实例会以较低的总成本、花费很长时间来完成这项工作。

支付不超过该结果的值

您需要进行数据处理工作。您将会对该工作的结果有一个很好的了解，以便于能够让您知道在计算成本方面它们的价值。

当您分析了实例类型的 Spot 价格历史记录之后，选择一个计算时间成本不高于该工作结果成本的价格。由于 Spot 价格的波动，该价格可能会达到或低于您的请求，所以您要创建一个持久请求，并允许它间歇运行。

快速获取计算容量

您对附加容量有一个无法预料的短期需求，该容量不能通过按需实例获取。当您分析了实例类型的 Spot 价格历史记录之后，您选择高于历史最高价格的价格，以大幅提高完成您请求的可能性，并继续计算，直到完成实例。

收集所需内容并选择策略后，就可以请求竞价型实例了。对于本教程，默认的最高 Spot 实例价格设置为与按需价格相同（本教程为 0.003 美元）。以这种方式设置价格可最大限度地提高请求被执行的机会。

创建竞价型实例请求

以下代码片段向您展示了如何使用之前收集的元素创建竞价型实例请求。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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];
}
```

此方法返回的重要值是竞价型实例请求 ID，它包含在返回的 [SpotInstanceRequest](#) 对象的 `SpotInstanceRequestId` 成员中。

Note

您需要为启动的任何竞价型实例付费。为避免不必要的开支，请务必[取消所有请求](#)并[终止所有实例](#)。

确定您的竞价型实例请求的状态

以下代码片段向您展示了如何获取有关您竞价型实例请求的信息。您可以使用这些信息在代码中做出某些决定，例如是否继续等待竞价型实例请求得到满足。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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];
}
```

```
}
```

该方法返回有关竞价型实例请求的信息，例如实例 ID、其状态和状态代码。有关竞价型实例请求的状态代码，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

清理您的竞价型实例请求

当您不再需要请求竞价型实例时，请务必取消任何未处理的请求，以防止这些请求被重新执行。以下代码片段演示了如何取消一个竞价型实例请求。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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);  
}
```

清理您的竞价型实例

为避免不必要的费用，终止任何从竞价型实例请求启动的实例非常重要；只是取消竞价型实例请求并不会终止您的实例，这意味着您需要继续为它们支付费用。以下代码片段演示了在获取活动竞价型实例的实例标识符后如何终止实例。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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");
    }
}
}
```

完整代码

以下代码示例调用前面描述的方法来创建和取消竞价型实例请求并终止竞价型实例。

SDK 参考

NuGet 程序包：

- [AWSSDK.EC2](#)

编程元素：

- 命名空间 [Amazon.EC2](#)

类 [AmazonEC2Client](#)

类 [InstanceType](#)

- 命名空间 [Amazon.EC2.Model](#)

类 [CancelSpotInstanceRequestsRequest](#)

类 [DescribeSpotInstanceRequestsRequest](#)

类 [DescribeSpotInstanceRequestsResponse](#)

类 [InstanceStateChange](#)

类 [LaunchSpecification](#)

类 [RequestSpotInstancesRequest](#)

类 [RequestSpotInstancesResponse](#)

类 [SpotInstanceRequest](#)

类 [TerminateInstancesRequest](#)

类 [TerminateInstancesResponse](#)

代码

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2SpotInstanceRequests
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Some default values.
            // These could be made into command-line arguments instead.
            var instanceType = InstanceType.T1Micro;
            string securityGroupName = "default";
            string spotPrice = "0.003";
            int instanceCount = 1;

            // Parse the command line arguments
            if((args.Length != 1) || (!args[0].StartsWith("ami-")))
            {
                Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
            }
        }
    }
}
```

```
        Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
Instances.");
        return;
    }

    // Create the Amazon EC2 client.
    var ec2Client = new AmazonEC2Client();

    // Create the Spot Instance request and record its ID
    Console.WriteLine("\nCreating spot instance request...");
    var req = await CreateSpotInstanceRequest(
        ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
    string requestId = req.SpotInstanceRequestId;

    // Wait for an EC2 Spot Instance to become active
    Console.WriteLine(
        $"Waiting for Spot Instance request with ID {requestId} to become active...");
    int wait = 1;
    var start = DateTime.Now;
    while(true)
    {
        Console.Write(".");

        // Get and check the status to see if the request has been fulfilled.
        var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
        if(requestInfo.Status.Code == "fulfilled")
        {
            Console.WriteLine($"Spot Instance request {requestId} " +
                $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
            break;
        }

        // Wait a bit and try again, longer each time (1, 2, 4, ...)
        Thread.Sleep(wait);
        wait = wait * 2;
    }

    // Show the user how long it took to fulfill the Spot Instance request.
    TimeSpan span = DateTime.Now.Subtract(start);
    Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

    // Perform actions here as needed.
    // For this example, simply wait for the user to hit a key.
    // That gives them a chance to look at the EC2 console to see
```

```
// the running instance if they want to.
Console.WriteLine("Press any key to start the cleanup...");
Console.ReadKey(true);

// Cancel the request.
// Do this first to make sure that the request can't be re-fulfilled
// once the Spot Instance has been terminated.
Console.WriteLine("Canceling Spot Instance request...");
await CancelSpotInstanceRequest(ec2Client, requestId);

// Terminate the Spot Instance that's running.
Console.WriteLine("Terminating the running Spot Instance...");
await TerminateSpotInstance(ec2Client, requestId);

Console.WriteLine("Done. Press any key to exit...");
Console.ReadKey(true);
}

//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,
    InstanceType instanceType, string spotPrice, int instanceCount)
{
    var launchSpecification = new LaunchSpecification{
        ImageId = amiId,
        InstanceType = instanceType
    };
    launchSpecification.SecurityGroups.Add(securityGroupName);
    var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
    };

    RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
    return result.SpotInstanceRequests[0];
}

//
// Method to get information about a Spot Instance request, including the status,
```

```
// instance ID, etc.
// It gets the information for a specific request (as opposed to all requests).
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
    return describeResponse.SpotInstanceRequests[0];
}

//
// Method to cancel a Spot Instance request
private static async Task CancelSpotInstanceRequest(
    IAmazonEC2 ec2Client, string requestId)
{
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(requestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}

//
// Method to terminate a Spot Instance
private static async Task TerminateSpotInstance(
    IAmazonEC2 ec2Client, string requestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(requestId);

    // Retrieve the Spot Instance request to check for running instances.
    DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

    // If there are any running instances, terminate them
    if( (describeResponse.SpotInstanceRequests[0].Status.Code
        == "request-canceled-and-instance-running")
        || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
    {
```

```
TerminateInstancesResponse response =
    await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
        InstanceIds = new List<string>(){
            describeResponse.SpotInstanceRequests[0].InstanceId } });
foreach (InstanceStateChange item in response.TerminatingInstances)
{
    Console.WriteLine($"\\n  Terminated instance: {item.InstanceId}");
    Console.WriteLine($"  Instance state: {item.CurrentState.Name}\\n");
}
}
}
}
```

其他注意事项

- 运行本教程后，最好登录 [Amazon EC2 控制台](#)，验证[竞价型实例请求](#)是否已取消，[竞价型实例](#)是否已终止。

使用AWS SDK for .NET访问 AWS Identity and Access Management (IAM)

AWS SDK for .NET支持 [AWS Identity and Access Management](#)，这是一项 Web 服务，能够确保 AWS 客户管理 AWS 中的用户以及用户权限。

AWS Identity and Access Management (IAM) 用户是您在 AWS 中创建的实体。实体代表与 AWS 交互的个人或应用程序。有关 IAM 用户的更多信息，请参阅《IAM 用户指南》中的 [IAM 用户](#) 以及 [IAM 和 STS 限制](#)。

您通过创建 IAM 策略向用户授予权限。策略包含一个策略文档，其中列出了用户可以执行的操作以及这些操作会影响的资源。有关 IAM 策略的更多信息，请参阅《IAM 用户指南》中的 [策略和权限](#)。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

API

AWS SDK for .NET为 IAM 客户端提供了 API。这些 API 使您能够使用用户、角色和访问密钥等 IAM 功能。

本节包含少量示例，向您展示使用这些 API 时可以遵循的模式。要查看完整的 API 集，请参阅[AWS SDK for .NET API 参考](#)（并滚动至“Amazon.IdentityManagement”）。

本节还包含[一个示例](#)，向您展示如何将 IAM 角色附加到 Amazon EC2 实例，以便更轻松地管理凭证。

IAM API 由 [AWSSDK.IdentityManagement](#) NuGet 程序包提供。

先决条件

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

主题

主题

- [通过 JSON 创建 IAM 托管式策略](#)
- [显示 IAM 托管式策略的策略文档](#)
- [使用 IAM 角色授予访问权限](#)

通过 JSON 创建 IAM 托管式策略

此示例向您展示如何使用AWS SDK for .NET根据给定的 JSON 策略文档创建 [IAM 托管式策略](#)。该应用程序创建 IAM 客户端对象，从文件中读取策略文档，然后创建策略。

Note

有关 JSON 格式的示例策略文档，请参阅本主题末尾的[其它注意事项](#)。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [创建策略](#)

- [完整代码](#)
- [其他注意事项](#)

创建策略

以下代码片段使用给定名称和策略文档创建一个 IAM 托管式策略。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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)});  
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.IdentityManagement](#)

编程元素：

- 命名空间 [Amazon.IdentityManagement](#)
 - 类 [AmazonIdentityManagementServiceClient](#)
- 命名空间 [Amazon.IdentityManagement.Model](#)
 - 类 [CreatePolicyRequest](#)
 - 类 [CreatePolicyResponse](#)

代码

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
    // = = = = =
    // Class to create an IAM policy with a given policy document
    class Program
    {
        private const int MaxArgs = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
            var parsedArgs = CommandLine.Parse(args);
            if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
            {
                PrintHelp();
                return;
            }

            // Get the application arguments from the parsed list
            string policyName =
                CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-name");
            string policyFilename =
                CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
            if( string.IsNullOrEmpty(policyName)
                || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
                CommandLine.ErrorExit(
                    "\nOne or more of the required arguments is missing or incorrect." +
                    "\nRun the command with no arguments to see help.");

            // Create an IAM service client
            var iamClient = new AmazonIdentityManagementServiceClient();

            // Create the new policy
```

```

    var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
    Console.WriteLine($"Policy {response.Policy.PolicyName} has been created.");
    Console.WriteLine($"  Arn: {response.Policy.Arn}");
}

//
// Method to create an IAM policy from a JSON file
private static async Task<CreatePolicyResponse> CreateManagedPolicy(
    IAmazonIdentityManagementService iamClient, string policyName, string
jsonFilename)
{
    return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
document.");
}
}

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

```

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

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

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

    return parsedArgs;
}

//
// Method to get an argument from the parsed command-line arguments
//
// Parameters:
// - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
// - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
```

```
// - keys: An array of keys to look for in parsedArgs.
public static string GetArgument(
    Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
{
    string retval = null;
    foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
    return retval ?? defaultReturn;
}

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

其他注意事项

- 以下是示例策略文档，您可以将其复制到 JSON 文件中并用作此应用程序的输入：

```
{
  "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": "*"
}
]
```

- 您可以通过查看 [IAM 控制台](#) 来验证策略是否已创建。使用筛选策略下拉菜单，选择客户托管。在您不再需要策略时将其删除。
- 有关创建策略的更多信息，请参阅 [IAM 用户指南](#) 中的 [创建 IAM 策略](#) 和 [IAM JSON 策略参考](#)

显示 IAM 托管式策略的策略文档

此示例说明如何使用 AWS SDK for .NET 显示策略文档。该应用程序创建 IAM 客户端对象，找到给定 IAM 托管式策略的默认版本，然后以 JSON 格式显示策略文档。

以下各节提供了此示例的片段。此后显示了 [该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [查找默认版本](#)
- [显示策略文档](#)
- [完整代码](#)

查找默认版本

以下代码片段查找给定 IAM 策略的默认版本。

[本主题末尾](#) 的示例显示了此片段的使用情况。

```
//
// 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;
}
```

显示策略文档

以下代码片段以 JSON 格式显示了给定 IAM 策略的策略文档。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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)}");
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.IdentityManagement](#)

编程元素：

- 命名空间 [Amazon.IdentityManagement](#)
 - 类 [AmazonIdentityManagementServiceClient](#)
- 命名空间 [Amazon.IdentityManagement.Model](#)
 - 类 [GetPolicyVersionRequest](#)
 - 类 [GetPolicyVersionResponse](#)
 - 类 [ListPolicyVersionsRequest](#)
 - 类 [ListPolicyVersionsResponse](#)
 - 类 [PolicyVersion](#)

代码

```
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)}");
}
}
}
```

使用 IAM 角色授予访问权限

本教程介绍如何使用AWS SDK for .NET在 Amazon EC2 实例上启用 IAM 角色。

概述

必须通过使用由 AWS 颁发的凭证对发送到 AWS 的所有请求进行加密签名。因此，您需要通过某种策略为 Amazon EC2 实例上运行的应用程序管理凭证。您必须以安全的方式分发、存储和轮换这些凭证，并且能让应用程序访问这些凭证。

使用 IAM 角色，您可以有效地管理这些凭证。您可以创建 IAM 角色并使用应用程序所需的权限对其进行配置，然后将该角色附加到 EC2 实例。有关使用 IAM 角色的好处的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。另外，有关 [IAM 角色](#) 的信息，请参阅 IAM 用户指南。

对于使用AWS SDK for .NET构建的应用程序，当应用程序为 AWS 服务构造客户端对象时，该对象会搜索来自多个潜在来源的凭证。它搜索的顺序显示在[凭证和配置文件解析](#)中。

如果客户端对象在任何其它来源中找不到凭证，它会检索与配置为 IAM 角色且位于 EC2 实例元数据中的凭证具有相同权限的临时凭证。这些凭证用于从客户端对象调用 AWS。

关于本教程

按照本教程操作时，您可以使用AWS SDK for .NET (和其它工具) 启动附加了 IAM 角色的 Amazon EC2 实例，然后使用 IAM 角色的权限查看该实例上的应用程序。

主题

- [创建示例 Amazon S3 应用程序](#)
- [创建 IAM 角色](#)
- [启动 EC2 实例并指定 IAM 角色](#)
- [连接到 EC2 实例](#)
- [在 EC2 实例上运行示例应用程序](#)
- [清理](#)

创建示例 Amazon S3 应用程序

此示例应用程序从 Amazon S3 中检索对象。要运行示例应用程序，您需要：

- 包含文本文件的 Amazon S3 桶。
- 开发计算机上允许您访问桶的 AWS 凭证。

有关创建 Amazon S3 存储桶并上载对象的信息，请参阅[Amazon Simple Storage Service 用户指南](#)。有关 AWS 凭证的更多信息，请参阅[使用 AWS 配置软件开发工具包身份验证](#)。

使用以下代码创建 .NET Core 项目。然后在开发计算机上测试该应用程序。

Note

在您的开发计算机上，安装了 .NET Core 运行时系统，这使您无需发布即可运行应用程序。在本教程中稍后创建 EC2 实例时，您可以选择在实例上安装 .NET Core 运行时系统。这为您提供了类似的体验和较小的文件传输。

但是，您也可以选择不在此实例上安装 .NET Core 运行时系统。如果您选择此操作方案，则必须发布应用程序，以便在将其转移到实例时包含所有依赖关系。

SDK 参考

NuGet 程序包：

- [AWSSDK.S3](#)

编程元素：

- 命名空间 [Amazon.S3](#)

类 [AmazonS3Client](#)

- 命名空间 [Amazon.S3.Model](#)

类 [GetObjectResponse](#)

代码

```
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);
}
}
```

```
}
```

如果需要，可以临时删除在开发计算机上使用的凭证，以查看应用程序的响应情况。（但请务必在完成后再恢复凭证。）

创建 IAM 角色

创建具有合适权限以访问 Amazon S3 的 IAM 角色。

1. 打开 [IAM 控制台](#)。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 选择 AWS 服务，找到并选择 EC2，然后选择下一步: 权限。
4. 在附加权限策略下，找到并选择 AmazonS3ReadOnlyAccess。如果愿意，请查看该策略，然后选择下一步: 标签。
5. 如果需要，可以添加标签，然后选择下一步: 查看。
6. 键入角色的名称和描述，然后选择创建角色。请记住此名称，因为在启动 EC2 实例时会用到它。

启动 EC2 实例并指定 IAM 角色

使用先前创建的 IAM 角色启动 EC2 实例。您可以通过下列方式来执行此操作。

- 使用 EC2 控制台

按照[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中的指示启动实例。

按照向导进行操作时，您至少应访问配置实例详细信息页面，以便可以选择之前创建的 IAM 角色。

- 使用 AWS SDK for .NET

有关此操作的信息，请参阅[启动 Amazon EC2 实例](#)，包括该主题接近末尾处的[其他注意事项](#)。

要启动附加了 IAM 角色的 EC2 实例，IAM 用户的配置必须包含某些权限。有关所需权限的更多信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

连接到 EC2 实例

连接到 EC2 实例，以便您可将示例应用程序传输到该实例，然后运行该应用程序。您将需要包含用于启动实例的密钥对私有部分的文件；即 PEM 文件。

您可以按照[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)中的连接程序来执行此操作。当您连接时，请确保您可以将文件从开发计算机传输到您的实例。

如果您在 Windows 上使用 Visual Studio，也可以使用 Toolkit for Visual Studio 连接到实例。有关更多信息，请参阅《AWS Toolkit for Visual Studio 用户指南》中的[连接到 Amazon EC2 实例](#)。

在 EC2 实例上运行示例应用程序

1. 将应用程序从本地驱动器复制到您的实例。

传输哪些文件取决于您如何构建应用程序以及您的实例是否安装了 .NET Core 运行时系统。有关如何将文件传输至实例的信息，请参阅[适用于 Linux 实例的 Amazon EC2 用户指南](#)或[适用于 Windows 实例的 Amazon EC2 用户指南](#)。

2. 启动应用程序并验证其运行结果是否与开发计算机上的结果相同。
3. 验证应用程序是否使用 IAM 角色提供的凭证。
 - a. 打开 [Amazon EC2 控制台](#)。
 - b. 选择实例，然后通过操作、实例设置和附加/替换 IAM 角色分离 IAM 角色。
 - c. 再次运行该应用程序，看看它是否返回了授权错误。

清理

完成本教程后，如果您不再需要自己创建的 EC2 实例，请务必终止该实例以避免不必要的开支。您可以在 [Amazon EC2 控制台](#) 中或以编程方式执行此操作，如[终止 Amazon EC2 实例](#)中所述。您还可以删除为本教程创建的其它资源。其中可能包括 IAM 角色、EC2 密钥对和 PEM 文件、安全组等。

使用 Amazon Simple Storage Service Internet 存储

AWS SDK for .NET 支持 ([I](#))，这是 Internet 上的存储。该服务旨在降低开发人员进行网络规模级计算的难度。

API

AWS SDK for .NET 为 Amazon S3 客户端提供了 API。这些 API 使您能够使用 Amazon S3 资源，例如桶和项目。要查看适用于 Amazon S3 的全套 API，请参阅以下内容：

- [AWS SDK for .NET API 参考](#) (然后滚动至“Amazon.S3”)。
- [Amazon.Extensions.S3.Encryption](#) 文档

Amazon S3 API 由以下 NuGet 程序包提供：

- [AWSSDK.S3](#)
- [Amazon.Extensions.S3.Encryption](#)

先决条件

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

本文档中的示例

本文档中的以下主题向您显示如何使用AWS SDK for .NET来利用 Amazon S3。

- [使用 KMS 密钥进行 S3 加密](#)

其它文档中的示例

以下指向 [Amazon S3 开发人员指南](#) 的链接提供了更多示例，说明如何使用 AWS SDK for .NET 来利用 Amazon S3。

Note

尽管这些示例和其它编程注意事项是为使用 .NET Framework 的AWS SDK for .NET版本 3 创建的，但它们也适用于使用 .NET Core 的AWS SDK for .NET的更高版本。有时需要对代码进行细微调整。

Amazon S3 编程示例

- [管理 ACL](#)
- [创建存储桶](#)
- [上传对象](#)
- [使用高级 API \(Amazon.S3.Transfer.TransferUtility \) 的多部分上传](#)

- [使用低级别 API 的分段上传](#)
- [列出对象](#)
- [列出密钥](#)
- [获取对象](#)
- [复制对象](#)
- [使用分段上传 API 复制对象](#)
- [删除对象](#)
- [删除多个对象](#)
- [恢复对象](#)
- [为通知配置存储桶](#)
- [管理对象的生命周期](#)
- [生成预签名对象 URL](#)
- [管理网站](#)
- [允许跨源资源共享 \(CORS\)](#)

其它编程注意事项

- [使用AWS SDK for .NET进行 Amazon S3 编程](#)
- [使用 IAM 用户临时证书创建请求](#)
- [使用联合身份用户临时凭证创建请求](#)
- [指定服务器端加密](#)
- [使用客户提供的加密密钥指定服务器端加密](#)

在 AWS SDK for .NET 中使用适用于 Amazon S3 加密的 AWS KMS 密钥

此示例说明如何使用 AWS Key Management Service 密钥对 Amazon S3 对象进行加密。该应用程序创建客户主密钥 (CMK) ，并使用它来创建用于客户端加密的 [AmazonS3EncryptionClientV2](#) 对象。应用程序使用该客户端根据现有 Amazon S3 桶中的给定文本文件创建加密对象。然后，它会解密对象并显示其内容。

⚠ Warning

名为AmazonS3EncryptionClient的类似类已被弃用，其安全性不如AmazonS3EncryptionClientV2类。要迁移使用AmazonS3EncryptionClient的现有代码，请参阅[S3 加密客户端迁移](#)。

主题

- [创建加密材料](#)
- [创建并加密 Amazon S3 对象](#)
- [完整代码](#)
- [其他注意事项](#)

创建加密材料

以下代码片段创建了一个包含 KMS 密钥 ID 的 EncryptionMaterials 对象。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
// 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);
```

创建并加密 Amazon S3 对象

以下代码片段创建了一个使用先前创建的加密材料的 AmazonS3EncryptionClientV2 对象。然后，它使用客户端来创建和加密一个新的 Amazon S3 对象。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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
    });
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [Amazon.Extensions.S3.Encryption](#)

编程元素：

- 命名空间 [Amazon.Extensions.S3.Encryption](#)

类 [AmazonS3EncryptionClientV2](#)

类 [AmazonS3CryptoConfigurationV2](#)

类 [CryptoStorageMode](#)

类 [EncryptionMaterialsV2](#)

- 命名空间 [Amazon.Extensions.S3.Encryption.Primitives](#)

类 [KmsType](#)

- 命名空间 [Amazon.S3.Model](#)

类 [GetObjectRequest](#)

类 [GetObjectResponse](#)

类 [PutObjectRequest](#)

- 命名空间 [Amazon.KeyManagementService](#)

类 [AmazonKeyManagementServiceClient](#)

- 命名空间 [Amazon.KeyManagementService.Model](#)

类 [CreateKeyRequest](#)

类 [CreateKeyResponse](#)

代码

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace KmsS3Encryption
{
    // = = = = =
    // Class to store text in an encrypted S3 object.
    class Program
    {
        private const int MaxArgs = 3;
```

```
public static async Task Main(string[] args)
{
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
    {
        PrintHelp();
        return;
    }

    // Get the application arguments from the parsed list
    string bucketName =
        CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
    string fileName =
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
    string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
    if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
            "\nOne or more of the required arguments is missing or incorrect." +
            "\nRun the command with no arguments to see help.");
    if(!File.Exists(fileName))
        CommandLine.ErrorExit($"The given file {fileName} doesn't exist.");
    if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

    // Create a customer master key (CMK) and store the result
    CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
    var kmsEncryptionContext = new Dictionary<string, string>();
    var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

    // Create the object in the bucket, then display the content of the object
    var putObjectResponse =
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
    Stream stream = putObjectResponse.ResponseStream;
    StreamReader reader = new StreamReader(stream);
    Console.WriteLine(reader.ReadToEnd());
}
```

```
//
// Method to create and encrypt an object in an S3 bucket
static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
    EncryptionMaterialsV2 materials, string bucketName,
    string fileName, string itemName)
{
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
    var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
    {
        StorageMode = CryptoStorageMode.ObjectMetadata
    };
    var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

    // Create, encrypt, and put the object
    await s3EncClient.PutObjectAsync(new PutObjectRequest
    {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
    });

    // Get, decrypt, and return the object
    return await s3EncClient.GetObjectAsync(new GetObjectRequest
    {
        BucketName = bucketName,
        Key = itemName
    });
}

//
// Command-line help
private static void PrintHelp()
{
    Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n -b, --bucket-name: The name of an existing S3 bucket." +
        "\n -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
}
}
```

```
}

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

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

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



```
        n++;
    }
}

return parsedArgs;
}

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

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

其他注意事项

- 您可以查看此示例的结果。为此，请前往 [Amazon S3 控制台](#) 并打开您提供给应用程序的桶。然后找到新对象，下载后在文本编辑器中打开该对象。

- [AmazonS3EncryptionClientV2](#) 类实现与标准 [AmazonS3Client](#) 类相同的接口。这样可以更轻松地将代码移植到 [AmazonS3EncryptionClientV2](#) 类中，从而在客户端中自动透明地进行加密和解密。
- 使用 AWS KMS 密钥作为主密钥的主要优势是无需存储和管理自己的主密钥，该工作由 AWS 来完成。第二个优势是 AWS SDK for .NET 的 [AmazonS3EncryptionClientV2](#) 类可与 AWS SDK for Java 的 [AmazonS3EncryptionClientV2](#) 类互操作。这意味着，您可以使用 AWS SDK for Java 加密，然后使用 AWS SDK for .NET 解密，反之亦然。

Note

只有在元数据模式中运行时，AWS SDK for .NET 的 [AmazonS3EncryptionClientV2](#) 类才支持 KMS 主密钥。AWS SDK for .NET 的 [AmazonS3EncryptionClientV2](#) 类的指令文件模式与 AWS SDK for Java 的 [AmazonS3EncryptionClientV2](#) 类不兼容。

- 有关使用 [AmazonS3EncryptionClientV2](#) 类进行客户端加密的更多信息以及信封加密的工作原理，请参阅[使用AWS SDK for .NET和 Amazon S3 进行客户端数据加密](#)。

使用 Amazon Simple Notification Service 从云端发送通知

Note

本主题中的信息特定于基于 .NET Framework 和 AWS SDK for .NET 3.3 版及更低版本的项目。

AWS SDK for .NET 支持 Amazon Simple Notification Service (Amazon SNS) ，这是一项 Web 服务，通过该服务，应用程序、终端用户和设备可以即时发送和接收来自云的通知。有关更多信息，请参阅 [Amazon SNS](#)。

列出您的 Amazon SNS 话题

以下示例说明如何列出您的 Amazon SNS 主题、每个主题的订阅以及每个主题的属性。本示例使用默认的 [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));
```

将消息发送到 Amazon SNS 主题

以下代码示例显示如何将消息发送到 Amazon SNS 主题中。该示例采用了一个参数，即 Amazon SNS 主题的 ARN。

```
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);
        }
    }
}
```

请参阅[完整示例](#)，包括有关在 GitHub 上如何从命令行构建并运行示例的信息。

向一个电话号码发送 SMS 消息

以下示例说明如何向电话号码发送 SMS 消息。该示例采用一个参数，即电话号码，该参数必须采用注释中描述的两种格式中的任何一种。

```
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);
}
}
}
```

请参阅[完整示例](#)，包括有关在 GitHub 上如何从命令行构建并运行示例的信息。

使用 Amazon SQS 发送消息

AWS SDK for .NET 支持 [Amazon Simple Queue Service \(Amazon SQS \)](#)，这是一个消息队列服务，可处理系统中组件之间的消息或工作流。

Amazon SQS 队列提供了一种机制，使您能够在微服务、分布式系统和无服务器应用程序等软件组件之间发送、存储和接收消息。这使您能够分离此类组件，无需设计和操作自己的消息传递系统。有关 Amazon SQS 中队列和消息的工作原理的信息，请参阅 [Amazon Simple Queue Service 开发人员指南](#) 中的 [Amazon SQS 教程](#) 和 [基本 Amazon SQS 架构](#)。

⚠ Important

由于队列的分布式特性，Amazon SQS 无法保证您以消息发送的准确顺序接收消息。如果您需要保留消息顺序，请使用 [Amazon SQS FIFO 队列](#)。

API

AWS SDK for .NET 为 Amazon SQS 客户端提供了 API。这些 API 使您能够使用队列和消息等 Amazon SQS 功能。本节包含少量示例，向您展示使用这些 API 时可以遵循的模式。要查看完整的 API 集，请参阅 [AWS SDK for .NET API 参考](#)（并滚动至“Amazon.SQS”）。

Amazon SQS API 由 [AWSSDK.SQS](#) NuGet 程序包提供。

先决条件

开始之前，请确保您已 [完成环境和项目的设置](#)。还要查看 [软件开发工具包功能](#) 中的信息。

主题

主题

- [创建 Amazon SQS 队列](#)
- [更新 Amazon SQS 队列](#)
- [删除 Amazon SQS 队列](#)
- [发送 Amazon SMS 消息](#)
- [接收 Amazon SQS 消息](#)

创建 Amazon SQS 队列

此示例向您展示如何使用 AWS SDK for .NET 创建 Amazon SQS 队列。如果您不提供 [死信队列](#) 的 ARN，则应用程序会创建一个死信队列。然后，它会创建一个标准消息队列，其中包括死信队列（您提供的队列或创建的队列）。

如果您不提供任何命令行参数，则应用程序仅显示有关所有现有队列的信息。

以下各节提供了此示例的片段。此后显示了 [该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [显示现有队列](#)
- [创建队列](#)
- [获取队列的 ARN](#)
- [完整代码](#)
- [其他注意事项](#)

显示现有队列

以下代码片段显示了 SQS 客户端区域中现有队列的列表以及每个队列的属性。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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}");
}
```

创建队列

以下代码片段创建队列。该片段包括死信队列的使用，但队列不一定需要死信队列。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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;
}
```

获取队列的 ARN

以下代码片段获取由给定队列 URL 标识的队列的 ARN。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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;
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.SQS](#)

编程元素：

- 命名空间 [Amazon.SQS](#)
 - 类 [AmazonSQSClient](#)
 - 类 [QueueAttributeName](#)
- 命名空间 [Amazon.SQS.Model](#)
 - 类 [CreateQueueRequest](#)
 - 类 [CreateQueueResponse](#)
 - 类 [GetQueueAttributesResponse](#)
 - 类 [ListQueuesResponse](#)

代码

```
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);
}
}
}
```

其他注意事项

- 您的队列名称必须由字母数字字符、连字符和下划线组成。
- 队列名称和队列 URL 区分大小写
- 如果您需要队列 URL 但只有队列名称, 请使用其中一种 `AmazonSQSClient.GetQueueUrlAsync` 方法。
- 有关您可以设置的各种队列属性的信息, 请参阅[AWS SDK for .NET API 参考](#)中的 [CreateQueueRequest](#) 或 [Amazon Simple Queue Service API 参考](#)中的 [SetQueueAttributes](#)。
- 此示例指定对您创建的队列中的所有消息进行长轮询。可使用 `ReceiveMessageWaitTimeSeconds` 属性执行此操作。

您还可以在调用 [AmazonSqsClient](#) 类的 `ReceiveMessageAsync` 方法期间指定长轮询。有关更多信息，请参阅[接收 Amazon SQS 消息](#)。

有关短轮询与长轮询的信息，请参阅《Amazon Simple Queue Service 开发人员指南》中的[短轮询与长轮询](#)。

- 其它（源）队列可将未成功处理的消息转到死信队列。有关更多信息，请参阅《Amazon Simple Queue Service 开发人员指南》中的 [Amazon SQS 死信队列](#)。
- 您还可以在 [Amazon SQS 控制台](#) 中查看队列列表和此示例的结果。

更新 Amazon SQS 队列

此示例向您展示如何使用 AWS SDK for .NET 更新 Amazon SQS 队列。经过一些检查后，应用程序使用给定值更新给定属性，然后显示队列的所有属性。

如果命令行参数中仅包含队列 URL，则应用程序仅显示队列的所有属性。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [显示队列属性](#)
- [验证属性名称](#)
- [更新队列属性](#)
- [完整代码](#)
- [其他注意事项](#)

显示队列属性

以下代码片段显示了由给定队列 URL 标识的队列的属性。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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}");
}
```

验证属性名称

以下代码片段验证了正在更新的属性的名称。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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;
}
```

更新队列属性

以下代码片段更新了由给定队列 URL 标识的队列属性。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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}});
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.SQS](#)

编程元素：

- 命名空间 [Amazon.SQS](#)

类 [AmazonSQSClient](#)

类 [QueueAttributeName](#)

- 命名空间 [Amazon.SQS.Model](#)

类 [GetQueueAttributesResponse](#)

代码

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
    // = = = = =
    // Class to update a queue
    class Program
    {
        private const int MaxArgs = 3;
        private const int InvalidArgCount = 2;

        static async Task Main(string[] args)
        {
            // Parse the command line and show help if necessary
```

```
var parsedArgs = CommandLine.Parse(args);
if(parsedArgs.Count == 0)
{
    PrintHelp();
    return;
}
if((parsedArgs.Count > MaxArgs) || (parsedArgs.Count == InvalidArgCount))
    CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
        "\nRun the command with no arguments to see help.");

// Get the application arguments from the parsed list
var qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

if(string.IsNullOrEmpty(qUrl))
    CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
        "\nRun the command with no arguments to see help.");

// Create the Amazon SQS client
var sqsClient = new AmazonSQSClient();

// In the case of one command-line argument, just show the attributes for the
queue
if(parsedArgs.Count == 1)
    await ShowAllAttributes(sqsClient, qUrl);

// Otherwise, attempt to update the given queue attribute with the given value
else
{
    // Check to see if the attribute is valid
    if(ValidAttribute(attribute))
    {
        // Perform the update and then show all the attributes of the queue
        await UpdateAttribute(sqsClient, qUrl, attribute, value);
        await ShowAllAttributes(sqsClient, qUrl);
    }
    else
    {
        Console.WriteLine($"The given attribute name, {attribute}, isn't valid.");
    }
}
}
```

```
//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
    GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
            new List<string>{ QueueAttributeName.All });
    Console.WriteLine($"Queue: {qUrl}");
    foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"{att.Key}: {att.Value}");
}

//
// Method to check the name of the attribute
private static bool ValidAttribute(string attribute)
{
    var attOk = false;
    var qAttNameType = typeof(QueueAttributeName);
    List<string> qAttNamefields = new List<string>();
    foreach(var field in qAttNameType.GetFields())
        qAttNamefields.Add(field.Name);
    foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
    return attOk;
}

//
// Method to update a queue attribute
private static async Task UpdateAttribute(
    IAmazonSQS sqsClient, string qUrl, string attribute, string value)
{
    await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
}

//
// Command-line help
private static void PrintHelp()
{
```

```

    Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
    Console.WriteLine("  -q: The URL of the queue you want to update.");
    Console.WriteLine("  -a: The name of the attribute to update.");
    Console.WriteLine("  -v, --value: The value to assign to the attribute.");
  }
}

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

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

```

```
    }

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

return parsedArgs;
}

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

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

其他注意事项

- 要更新 `RedrivePolicy` 属性，必须根据您的操作系统，引用整个值并对键/值对的引号进行转义。

例如，在 Windows 上，该值的构造方式类似于以下内容：

```
"{\"deadLetterTargetArn\": \"DEAD_LETTER-QUEUE-ARN\", \"maxReceiveCount\": \"10\"}"
```

删除 Amazon SQS 队列

此示例向您展示如何使用 AWS SDK for .NET 删除 Amazon SQS 队列。应用程序删除队列，等到队列消失，然后显示剩余队列的列表。

如果您不提供任何命令行参数，则应用程序仅显示现有队列的列表。

以下各节提供了此示例的片段。此后显示了 [该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [删除队列](#)
- [等待队列消失](#)
- [显示现有队列的列表](#)
- [完整代码](#)
- [其他注意事项](#)

删除队列

以下代码片段删除了由给定队列 URL 标识的队列。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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.");  
}
```


等待队列消失

以下代码片段等待删除过程完成，这可能需要 60 秒。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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;
    }
}
```

显示现有队列的列表

以下代码片段显示了 SQS 客户端区域中现有队列的列表。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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}");
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.SQS](#)

编程元素：

- 命名空间 [Amazon.SQS](#)
 - 类 [AmazonSQSClient](#)
- 命名空间 [Amazon.SQS.Model](#)
 - 类 [ListQueuesResponse](#)

代码

```
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}");
}
}
```

```
}
```

其他注意事项

- DeleteQueueAsync API 调用不会检查您要删除的队列是否被用作死信队列。可以用更复杂的程序来检查这一点。
- 您还可以在 [Amazon SQS 控制台](#) 中查看队列列表和此示例的结果。

发送 Amazon SMS 消息

此示例向您展示如何使用 AWS SDK for .NET 向 Amazon SQS 队列发送消息，您可以通过[编程方式](#)或使用 [Amazon SQS 控制台](#) 创建该队列。应用程序向队列发送一条消息，然后发送一批消息。然后，应用程序等待用户输入，这些输入可以是要发送到队列的额外消息或退出应用程序的请求。

此示例和[下一个有关接收消息的示例](#)可以一起使用，以查看 Amazon SQS 中的消息流。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [发送消息](#)
- [发送一批消息](#)
- [从队列中删除所有消息](#)
- [完整代码](#)
- [其他注意事项](#)

发送消息

以下代码片段将消息发送到由给定队列 URL 标识的队列。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//  
// 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}");
}
```

发送一批消息

以下代码片段向由给定队列 URL 标识的队列发送一批消息。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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.");
}
```

从队列中删除所有消息

以下代码片段删除了来自由给定队列 URL 标识的队列的所有消息。这也称为清除队列。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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}");
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.SQS](#)

编程元素：

- 命名空间 [Amazon.SQS](#)

类 [AmazonSQSClient](#)

- 命名空间 [Amazon.SQS.Model](#)

类 [PurgeQueueResponse](#)

类 [SendMessageBatchResponse](#)

类 [SendMessageResponse](#)

类 [SendMessageBatchRequestEntry](#)

类 [SendMessageBatchResultEntry](#)

代码

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
    // = = = = =
    // Class to send messages to a queue
    class Program
    {
```

```
// Some example messages to send to the queue
private const string JsonMessage = "{\"product\": [{\"name\": \"Product A\", \"price\": \"32\"}, {\"name\": \"Product B\", \"price\": \"27\"}]}";
private const string XmlMessage = "<products><product name=\\\"Product A\\\" price=\\\"32\\\" /><product name=\\\"Product B\\\" price=\\\"27\\\" /></products>";
private const string CustomMessage = "||product|Product A|32||product|Product B|27||";
private const string TextMessage = "Just a plain text message.";

static async Task Main(string[] args)
{
    // Do some checks on the command-line
    if(args.Length == 0)
    {
        Console.WriteLine("\nUsage: SQSSendMessages queue_url");
        Console.WriteLine("    queue_url - The URL of an existing SQS queue.");
        return;
    }
    if(!args[0].StartsWith("https://sqs."))
    {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
    }

    // Create the Amazon SQS client
    var sqsClient = new AmazonSQSClient();

    // (could verify that the queue exists)
    // Send some example messages to the given queue
    // A single message
    await SendMessage(sqsClient, args[0], JsonMessage);

    // A batch of messages
    var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
    await SendMessageBatch(sqsClient, args[0], batchMessages);

    // Let the user send their own messages or quit
    await InteractWithUser(sqsClient, args[0]);

    // Delete all messages that are still in the queue
}
```



```
    await DeleteAllMessages(sqsClient, args[0]);
}

//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
{
    SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}

//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
    Console.WriteLine($"Sending a batch of messages to queue\n {qUrl}");
    SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
}

//
// Method to get input from the user
// They can provide messages to put in the queue or exit the application
private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
{
    string response;
    while (true)
    {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
    }
}
```

```
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
    }
}

//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
    Console.WriteLine($"\\nPurging messages from queue\\n {qUrl}...");
    PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
    Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
}
}
```

其他注意事项

- 有关消息的各种限制 (包括允许的字符) 的信息，请参阅 [Amazon Simple Queue Service 开发人员指南中与消息相关的配额](#)部分。
- 消息会一直保留在队列中，直到被删除或队列被清除。当应用程序收到一条消息时，即使它仍然存在于队列中，它也不会出现在队列中。有关可见性超时的更多信息，请参阅 [Amazon SQS 可见性超时](#)。
- 除了消息正文外，您还可以为消息添加属性。有关更多信息，请参阅[消息元数据](#)。

接收 Amazon SQS 消息

此示例向您展示如何使用AWS SDK for .NET从 Amazon SQS 队列接收消息，您可以通过[编程方式](#)或使用 [Amazon SQS 控制台](#)创建该队列。应用程序从队列中读取一条消息，处理该消息 (在本例中，在控制台上显示消息正文)，然后从队列中删除该消息。应用程序会重复这些步骤，直到用户在键盘上键入一个键。

此示例和[前面有关接收消息的示例](#)可以一起使用，以查看 Amazon SQS 中的消息流。

以下各节提供了此示例的片段。此后显示了[该示例的完整代码](#)，并且可以按原样构建和运行。

主题

- [接收消息](#)
- [删除消息](#)
- [完整代码](#)
- [其他注意事项](#)

接收消息

以下代码片段从由给定队列 URL 标识的队列接收消息。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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.)
    });
}
```

删除消息

以下代码片段删除了来自由给定队列 URL 标识的队列的消息。

[本主题末尾](#)的示例显示了此片段的使用情况。

```
//
// 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);
}
```

完整代码

本部分显示了本示例的相关参考和完整代码。

SDK 参考

NuGet 程序包：

- [AWSSDK.SQS](#)

编程元素：

- 命名空间 [Amazon.SQS](#)
 - 类 [AmazonSQSClient](#)
- 命名空间 [Amazon.SQS.Model](#)
 - 类 [ReceiveMessageRequest](#)
 - 类 [ReceiveMessageResponse](#)

代码

```
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);
}
}
```

其他注意事项

- 为了指定长轮询，此示例在每次调用 `ReceiveMessageAsync` 方法时都使用该 `WaitTimeSeconds` 属性。

您还可以在[创建](#)或[更新](#)队列时使用 `ReceiveMessageWaitTimeSeconds` 属性为队列中的所有消息指定长轮询。

有关短轮询与长轮询的信息，请参阅《Amazon Simple Queue Service 开发人员指南》中的[短轮询与长轮询](#)。

- 在消息处理过程中，您可以使用接收句柄来更改消息可见性超时。有关如何执行此操作的信息，请参阅 [AmazonSQSClient](#) 类的 `ChangeMessageVisibilityAsync` 方法。
- 调用 `DeleteMessageAsync` 方法将无条件地从队列中删除消息，而无论可见性超时设置如何。

将 AWS Lambda 用于计算服务

AWS SDK for .NET 支持 AWS Lambda，因此您可以运行代码，而无需预置或管理服务器。有关更多信息，请参阅 [AWS Lambda 产品页面](#) 和 [AWS Lambda 开发人员指南](#)，尤其是 [使用 C#](#) 部分。

API

AWS SDK for .NET 为 AWS Lambda 提供了 API。这些 API 使您能够使用 [函数](#)、[触发器](#) 和 [事件](#) 等 Lambda 功能。要查看完整的 API 集，请参阅 [AWS SDK for .NET API 参考](#) 中的 [Lambda](#)。

Lambda API 由 [NuGet 程序包提供](#)。

先决条件

开始之前，请确保您已 [完成环境和项目的设置](#)。还要查看 [软件开发工具包功能](#) 中的信息。

主题

主题

- [使用注解来编写 AWS Lambda 函数](#)

使用注解来编写 AWS Lambda 函数

在编写 Lambda 函数时，您有时需要编写大量的处理程序代码，并更新 AWS CloudFormation 模板以及执行其它任务。Lambda 注释是一个框架，有助于减轻 .NET 6 Lambda 函数的负担，从而让使用 C# 编写 Lambda 的体验更加自然。

如果要举例说明使用 Lambda 注释框架的好处，请考虑以下将两个数字相加的代码片段。

不使用 Lambda 注释

```
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" } }
    };
}
}
```

使用 Lambda 注释

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

如示例所示，Lambda 注释可以消除对某些 Boilerplate 代码的需求。

有关如何使用该框架以及更多信息，请参阅以下资源：

- [GitHub 自述](#)文件，其中包含有关 Lambda 注释 API 和属性的文档。
- 有关 Lambda 注释的[博客文章](#)。
- [Amazon.Lambda.Annotations](#) NuGet 程序包。

- GitHub 上的[照片资产管理项目](#)。具体而言，请参阅项目[自述文件](#)中的 [PamApiAnnotations](#) 文件夹和对 Lambda 注释的引用。

AWS SDK for .NET 的高级库和框架

以下各节包含有关不属于 SDK 核心功能的高级库和框架的信息。这些库和框架使用核心 SDK 功能来创建可简化某些任务的功能。

如果您不熟悉AWS SDK for .NET，则可能需要先查看[快速了解](#)主题。其中提供了对 SDK 的简单介绍。

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

主题

- [AWS Message Processing Framework for .NET](#)

AWS Message Processing Framework for .NET

这是适用于预览版中功能的预发布文档。本文档随时可能更改。

.NET AWS 消息处理框架是一个AWS原生框架，它简化了使用亚马逊简单队列服务 (SQS)、亚马逊简单通知AWS服务 (SNS) Simple Notification Service 和亚马逊等服务的.NET 消息处理应用程序的开发。EventBridge该框架减少了开发人员需要编写的样板代码量，使您能够在发布和使用消息时专注于业务逻辑。

有关 Message Processing Framework 以及如何使用它的更多信息，请参阅以下资源。

- GitHub 存储库中的README文件，网址为 <https://github.com/awslabs/aws-dotnet-messaging>。
- 开发者文档[网址为 https://awslabs.github.io/aws-dotnet-messaging/docs/overview.html](https://awslabs.github.io/aws-dotnet-messaging/docs/overview.html)。

开始之前，请确保您已[完成环境和项目的设置](#)。还要查看[软件开发工具包功能](#)中的信息。

对 AWS OpsWorks 进行编程以使用堆栈和应用程序

Warning

AWS OpsWorks 生命周期即将终止，不再接受新客户。现有客户在 2024 年 3 月或 5 月之前不会受到影响，具体取决于他们使用的服务，届时该服务将不再可用。为了准备这种转换，建议现有客户尽快迁移到其他解决方案。有关更多信息，请参阅 [OpsWorks 产品页](#)。

AWS SDK for .NET 支持 AWS OpsWorks，后者提供了一种用于创建和管理堆栈及应用程序的简单灵活的方式。利用 AWS OpsWorks，您可以预配置 AWS 资源、管理其配置、向这些资源部署应用程序以及监控其运行状况。有关更多信息，请参阅 [OpsWorks 产品页面](#) 和 [AWS OpsWorks 用户指南](#)。

API

AWS SDK for .NET 为 AWS OpsWorks 提供了 API。这些 API 使您能够使用诸如 [堆栈及其层](#)、[实例](#) 和 [应用程序](#) 之类的 AWS OpsWorks 功能。要查看完整的 API 集，请参阅 [AWS SDK for .NET API 参考](#)（并滚动至“Amazon.OpsWorks”）。

AWS OpsWorks API 由 [AWSSDK.OpsWorks](#) NuGet 程序包提供。

先决条件

开始之前，请确保您已 [完成环境和项目的设置](#)。还要查看 [软件开发工具包功能](#) 中的信息。

对其它 AWS 服务和配置的支持

除前面各部分中描述的服务外，AWS SDK for .NET 还支持 AWS 服务。有关所有支持服务的 API 的信息，请参阅 [AWS SDK for .NET API 参考](#)。

除了各个 AWS 服务的命名空间之外，AWS SDK for .NET 还提供了以下 API：

区域图	描述	资源
AWS Support	以编程方式访问 AWS Support 案例和 Trusted Advisor 功能。	请参阅 Amazon.AWSSupport 和 Amazon.AWSSupport.Model 。

区域图	描述	资源
常规	帮助程序类和枚举。	请参阅 Amazon 和 Amazon.Util 。

AWS SDK for .NET 代码示例

本主题中的代码示例向您展示了如何将AWS SDK for .NET 与 AWS 结合使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

跨服务示例是指跨 AWS 服务 运行的示例应用程序。

示例

- [使用 AWS SDK for .NET 的操作和场景](#)
- [使用 AWS SDK for .NET 的跨服务示例](#)

使用 AWS SDK for .NET 的操作和场景

以下代码示例演示如何将 AWS SDK for .NET 与 AWS 服务结合使用来执行操作和实施常见场景。

操作 是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

服务

- [使用 AWS SDK for .NET 的 ACM 示例](#)
- [使用 AWS SDK for .NET 的 Aurora 示例](#)
- [使用 AWS SDK for .NET 的 Auto Scaling 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Bedrock 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Bedrock 运行时系统示例](#)
- [CloudWatch 使用示例 AWS SDK for .NET](#)
- [CloudWatch 使用记录示例 AWS SDK for .NET](#)
- [使用 AWS SDK for .NET 的 Amazon Cognito 身份提供商示例](#)
- [使用 AWS SDK for .NET 的 Amazon Comprehend 示例](#)
- [使用 AWS SDK for .NET 的 DynamoDB 示例](#)
- [使用 AWS SDK for .NET 的 Amazon EC2 示例](#)

- [使用 AWS SDK for .NET 的 Amazon ECS 示例](#)
- [使用 AWS SDK for .NET 的 Elastic Load Balancing 示例](#)
- [EventBridge 使用示例 AWS SDK for .NET](#)
- [使用 AWS SDK for .NET 的 AWS Glue 示例](#)
- [使用 AWS SDK for .NET 的 IAM 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Keyspaces 示例](#)
- [使用 AWS SDK for .NET 的 Kinesis 示例](#)
- [使用 AWS SDK for .NET 的 AWS KMS 示例](#)
- [使用 AWS SDK for .NET 的 Lambda 示例](#)
- [MediaConvert 使用示例 AWS SDK for .NET](#)
- [使用 AWS SDK for .NET 的 Organizations 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Polly 示例](#)
- [使用 AWS SDK for .NET 的 Amazon RDS 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Rekognition 示例](#)
- [使用 AWS SDK for .NET 的 Route 53 域注册示例](#)
- [使用 AWS SDK for .NET 的 Amazon S3 示例](#)
- [使用 AWS SDK for .NET 的 S3 Glacier 示例](#)
- [SageMaker 使用示例 AWS SDK for .NET](#)
- [使用 AWS SDK for .NET 的 Secrets Manager 示例](#)
- [使用 AWS SDK for .NET 的 Amazon SES 示例](#)
- [使用 AWS SDK for .NET 的 Amazon SNS 示例](#)
- [使用 AWS SDK for .NET 的 Amazon SQS 示例](#)
- [使用 AWS SDK for .NET 的 Step Functions 示例](#)
- [使用 AWS SDK for .NET 的 AWS STS 示例](#)
- [使用 AWS SDK for .NET 的 AWS Support 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Transcribe 示例](#)
- [使用 AWS SDK for .NET 的 Amazon Translate 示例](#)

使用 AWS SDK for .NET 的 ACM 示例

以下代码示例演示如何将 AWS SDK for .NET 与 ACM 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

描述证书

以下代码示例演示如何描述 ACM 证书。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeCertificate](#) 中的。

列出证书

以下代码示例演示如何列出 ACM 证书。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListCertificates](#) 中的。

使用 AWS SDK for .NET 的 Aurora 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Aurora 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Aurora

以下代码示例显示如何开始使用 Aurora。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}.");
        }
    }
}
```

```
    }  
  }  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBClusters](#)。

主题

- [操作](#)
- [场景](#)

操作

创建数据库集群

以下代码示例演示如何创建 Aurora 数据库集群。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
    }  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [CreateDBCluster](#)。

创建数据库集群参数组

以下代码示例演示如何创建 Aurora 数据库集群参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }

```

- 有关 API 的详细信息，请参阅 [AWS SDK for .NET API 参考 ClusterParameterGroup](#) 中的 [CreateDB](#)。

创建数据库集群快照

以下代码示例演示如何创建 Aurora 数据库集群快照。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

    /// <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;
    }

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ClusterSnapshot 中的 [CreateDB](#)。

在数据库集群中创建数据库实例

以下代码示例演示如何在 Aurora 数据库集群中创建数据库实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

    /// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [CreateDBInstance](#)。

删除数据库集群

以下代码示例演示如何删除 Aurora 数据库集群。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DeleteDBCluster](#)。

删除数据库集群参数组

以下代码示例演示如何删除 Aurora 数据库集群参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <summary>
/// Delete a particular parameter group by name.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterParameterGroupNameAsync(string groupName)
{
    var request = new DeleteDBClusterParameterGroupRequest
    {
        DBClusterParameterGroupName = groupName,
    };

    var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
}
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ClusterParameterGroup](#) 中的 [deletedB](#)。

删除数据库实例

以下代码示例演示如何删除 Aurora 数据库实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DeleteDBInstance](#)。

描述数据库集群参数组

以下代码示例演示如何描述 Aurora 数据库集群参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ClusterParameterGroups](#) 中的 [describedB](#)。

描述数据库集群快照

以下代码示例演示如何描述 Aurora 数据库集群快照。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ClusterSnapshots 中的 [describeDB](#)。

描述数据库集群

以下代码示例演示如何描述 Aurora 数据库集群。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBClusters](#)。

描述数据库实例

以下代码示例演示如何描述 Aurora 数据库实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBInstances](#)。

描述数据库引擎版本

以下代码示例演示如何描述 Aurora 数据库引擎版本。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 EngineVersions 中的 [describedB](#)。

描述数据库实例的选项

以下代码示例演示如何描述 Aurora 数据库实例的选项。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 InstanceOptions 中的 [DescribeOrderable 数据库](#)。

描述数据库集群参数组中的参数

以下代码示例演示如何描述 Aurora 数据库集群参数组中的参数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 [AWS SDK for .NET API 参考 ClusterParameters 中的 `DescribeB`](#)。

更新数据库集群参数组中的参数

以下代码示例演示如何更新 Aurora 数据库集群参数组中的参数。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 的详细信息，请参阅 [AWS SDK for .NET API 参考 ClusterParameterGroup](#) 中的 [modifyDB](#)。

场景

开始使用数据库集群

以下代码示例演示了操作流程：

- 创建自定义 Aurora 数据库集群参数组并设置参数值。
- 创建一个使用参数组的数据库集群。
- 创建包含数据库的数据库实例。
- 拍摄数据库集群的快照，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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(
            $"{p.ParameterName}." +
            $"{p.Description}." +
            $"{p.AllowedValues}." +
            $"{p.ParameterValue}."));

    Console.WriteLine(sepBar);
}

/// <summary>
/// Choose a DB engine version.
/// </summary>
/// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
/// <returns>The selected engine version.</returns>
```

```
public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
{
    Console.WriteLine(sepBar);
    // Get a list of allowed engines.
    var allowedEngines =
        await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

    Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
    int i = 1;
    foreach (var version in allowedEngines)
    {
        Console.WriteLine(
            $"{i}. 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>
/// 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(
            $"{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(
        $"{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($"Clean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
    {
        // Delete the DB instance.
        Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
        await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
    }

    if (newCluster is not null && GetYesNoResponse($"Clean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
    {
        // Delete the DB cluster.
        Console.WriteLine($"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.WriteLine(".");
            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;
    }
}

```

场景调用以管理 Aurora 操作的包装程序方法。

```

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> DeleteClusterParameterGroupNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };
    }
}
```

```

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
    instance.</param>

```

```
/// <returns>List of DB instances.</returns>
public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
{
    var results = new List<DBInstance>();
    var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
        new DescribeDBInstancesRequest
        {
            DBInstanceIdentifier = dbInstanceIdentifier
        });
    // Get the entire list using the paginator.
    await foreach (var instances in instancesPaginator.DBInstances)
    {
        results.Add(instances);
    }
    return results;
}

/// <summary>
/// Returns a list of DB clusters.
/// </summary>
/// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
/// <returns>List of DB clusters.</returns>
public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
{
    var results = new List<DBCluster>();

    DescribeDBClustersResponse response;
    DescribeDBClustersRequest request = new DescribeDBClustersRequest
    {
        DBClusterIdentifier = dbClusterIdentifier
    };
    // Get the full list if there are multiple pages.
    do
    {
        response = await _amazonRDS.DescribeDBClustersAsync(request);
        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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateDBCluster](#)
 - [创建数据库 ClusterParameterGroup](#)
 - [创建数据库 ClusterSnapshot](#)
 - [CreateDBInstance](#)
 - [DeleteDBCluster](#)
 - [已删除B ClusterParameterGroup](#)
 - [DeleteDBInstance](#)

- [describedB ClusterParameterGroups](#)
- [describedB ClusterParameters](#)
- [describedB ClusterSnapshots](#)
- [DescribeDBClusters](#)
- [describedB EngineVersions](#)
- [DescribeDBInstances](#)
- [DescribeOrderable数据库 InstanceOptions](#)
- [modifyDB ClusterParameterGroup](#)

使用 AWS SDK for .NET 的 Auto Scaling 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Auto Scaling 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Auto Scaling

以下代码示例演示如何开始使用 Auto Scaling。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAutoScalingGroups](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

将 ELB 目标组附加到 Auto Scaling 组

以下代码示例演示如何将 ELB 目标组附加到 Auto Scaling 组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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 }
        });
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AttachLoadBalancerTargetGroups](#) 中的。

创建组

以下代码示例演示了如何创建自动扩缩组。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateAutoScalingGroup](#) 中的。

删除组

以下代码示例演示了如何删除自动扩缩组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

将自动扩缩组的最小大小更新为零，终止该组中的所有实例，然后删除该组。

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



```
        });
        stopping = true;
    }
    catch (ScalingActivityInProgressException)
    {
        Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
        Thread.Sleep(10000);
    }
}

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

/// <summary>
```

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

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

```
/// <summary>
/// Delete an Auto Scaling group.
/// </summary>
/// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
```

```
public async Task<bool> DeleteAutoScalingGroupAsync(
    string groupName)
{
    var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        ForceDelete = true,
    };

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

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

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteAutoScalingGroup](#) 中的。

禁用组指标集合

以下代码示例说明如何禁用 Auto Scaling 组的 CloudWatch 指标收集。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DisableMetricsCollection](#) 中的。

启用组的指标收集

以下代码示例显示了如何为 Auto Scaling 组启用 CloudWatch 指标收集。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [EnableMetricsCollection](#) 中的。

获取有关组的信息

以下代码示例演示了如何获取有关自动扩缩组的信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAutoScalingGroups](#) 中的。

获取有关实例的信息

以下代码示例演示了如何获取有关自动扩缩实例的信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAutoScalingInstances](#) 中的。

获取有关扩缩活动的信息

以下代码示例演示了如何获取有关自动扩缩活动的信息。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeScalingActivities](#) 中的。

设置组的所需容量

以下代码示例演示了如何设置自动扩缩组的所需容量。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SetDesiredCapacity](#) 中的。

终止组中的实例

以下代码示例演示了如何终止自动扩缩组中的实例。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [TerminateInstanceInAutoScalingGroup](#) 中的。

更新组

以下代码示例演示了如何更新自动扩缩组的配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateAutoScalingGroup](#) 中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud (Amazon EC2) 实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 AWS Systems Manager 参数控制 Web 服务器对请求和运行状况检查的响应。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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);
                }
            }
        }
    }
}
```

```

        {
            await
            _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
            ipString);
        }
    }
    loadBalancerAccess = await
    _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

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

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

```

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

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

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

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

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

Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
```

```
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

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

```
        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

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

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

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

        if (interactive)
            await DemoActionChoices();

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

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"Even while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
```

```
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

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

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

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

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

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

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

```

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
            _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
            _recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

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

```

创建一个包含自动扩缩和 Amazon EC2 操作的类。

```

/// <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 = "{" +
        "\nVersion\": \"2012-10-17\", \" +
        "\nStatement\": [{\" +
            "\nEffect\": \"Allow\", \" +
            "\nPrincipal\": {\" +
            "\nService\": [\" +
                "\nec2.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 = deleteKeyName });
    File.Delete($"{deleteKeyName}.pem");
}
catch (FileNotFoundException)
{
    Console.WriteLine($"Key pair {deleteKeyName} not found.");
}
}

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

```

创建一个包含弹性负载均衡操作的类。

```

/// <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);
        }
    }
}
}

```

创建一个使用 DynamoDB 模拟推荐服务的类。

```

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

```

```
public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

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

```
        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

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

var request = new DescribeTableRequest
{
    TableName = tableName
};

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

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

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

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

```
        return false;
    }
}

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

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

    await batchWrite.ExecuteAsync();
}

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

```
}  
}
```

创建一个包含 Systems Manager 操作的类。

```
/// <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 });
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)

- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacesIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

管理组和实例

以下代码示例演示了如何：

- 创建包含启动模板和可用区的 Amazon EC2 Auto Scaling 组，并获取有关正在运行的实例的信息。
- 启用 Amazon CloudWatch 指标收集。
- 更新组的所需容量，并等待实例启动。
- 终止组中的实例。
- 列出为响应用户请求和容量变化而发生的扩缩活动。
- 获取 CloudWatch 指标的统计数据，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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($"Metrics collected for {groupName}:");
    metrics.ForEach(metric =>
    {
        Console.WriteLine($"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.WriteLine("\nPress <Enter> to continue. ");
    _ = Console.ReadLine();
    Console.WriteLine();
}

/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter">The string to be centered.</param>
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
```

```
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);

    // Wait for the requested number of seconds.
    for (int i = numSeconds; i > 0; i--)
    {
        System.Threading.Thread.Sleep(1000);
        Console.Write($"{i}...");
    }

    PressEnter();
}
}
```

定义场景调用以管理启动模板和指标的函数。这些函数包含 Auto Scaling、Amazon EC2 和 CloudWatch 操作。

```
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
```

```
        {
            LaunchTemplateName = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.WriteLine($"{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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAutoScalingInstances](#)
 - [DescribeScalingActivities](#)
 - [DisableMetricsCollection](#)

- [EnableMetricsCollection](#)
- [SetDesiredCapacity](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

使用 AWS SDK for .NET 的 Amazon Bedrock 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Bedrock 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon Bedrock

以下代码示例演示了如何开始使用 Amazon Bedrock。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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)}");
}
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFoundationModels](#) 中的。

主题

- [操作](#)

操作

列出可用的 Amazon Bedrock 基础模型

以下代码示例演示如何列出可用的 Amazon Bedrock 基础模型。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Bedrock 基础模型。

```
/// <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);
    }
}
```



```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFoundationModels](#) 中的。

使用 AWS SDK for .NET 的 Amazon Bedrock 运行时系统示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Bedrock 运行时系统结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)
- [场景](#)

操作

使用 AI21 Labs Jurassic-2 生成文本

以下代码示例演示了如何通过调用 Amazon Bedrock 上的 AI21 Labs Jurassic-2 模型来生成文本。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 AI21 Labs Jurassic-2 基础模型。

```
    /// <summary>
    /// Asynchronously invokes the AI21 Labs Jurassic-2 model to run an
    inference based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Claude to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
    formats.
    /// For the format, ranges, and default values for AI21 Labs Jurassic-2,
    refer to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-jurassic2.html
    /// </remarks>
    public static async Task<string> InvokeJurassic2Async(string prompt)
    {
        string jurassic2ModelId = "ai21.j2-mid-v1";

        AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

        string payload = new JsonObject()
        {
            { "prompt", prompt },
            { "maxTokens", 200 },
            { "temperature", 0.5 }
        }.ToJsonString();

        string generatedText = "";
        try
        {
            InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
            {
                ModelId = jurassic2ModelId,
                Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
                ContentType = "application/json",
                Accept = "application/json"
            });

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                return JsonNode.ParseAsync(response.Body)
                    .Result?["completions"]?

```

```
        .ToArray()[0]?["data"]?
        .AsObject()["text"]?.GetValue<string>() ?? "";
    }
    else
    {
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModel](#) 中的。

使用 Anthropic Claude 2 生成文本

以下代码示例演示了如何通过调用 Amazon Bedrock 上的 Anthropic Claude 2 模型来生成文本。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Anthropic Claude 2 基础模型来生成文本。

```
/// <summary>
/// Asynchronously invokes the Anthropic Claude 2 model to run an inference
based on the provided input.
/// </summary>
/// <param name="prompt">The prompt that you want Claude to complete.</
param>
/// <returns>The inference response from the model</returns>
```

```
    /// <remarks>
    /// The different model providers have individual request and response
formats.
    /// For the format, ranges, and default values for Anthropic Claude, refer
to:
    ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-claude.html
    /// </remarks>
public static async Task<string> InvokeClaudeAsync(string prompt)
{
    string claudeModelId = "anthropic.claude-v2";

    // Claude requires you to enclose the prompt as follows:
    string enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

    AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

    string payload = new JsonObject()
    {
        { "prompt", enclosedPrompt },
        { "max_tokens_to_sample", 200 },
        { "temperature", 0.5 },
        { "stop_sequences", new JSONArray("\n\nHuman:") }
    }.ToJsonString();

    string generatedText = "";
    try
    {
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
        {
            ModelId = claudeModelId,
            Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
            ContentType = "application/json",
            Accept = "application/json"
        });

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            return JsonNode.ParseAsync(response.Body).Result?
["completion"]?.GetValue<string>() ?? "";
        }
        else
        {
```

```
        Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine(e.Message);
}
return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModel](#) 中的。

使用 Meta Llama 2 Chat 生成文本

以下代码示例演示了如何通过调用 Amazon Bedrock 上的 Meta Llama 2 Chat 模型来生成文本。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Meta Llama 2 基础模型以生成文本。

```
    /// <summary>
    /// Asynchronously invokes the Meta Llama 2 Chat model to run an inference
based on the provided input.
    /// </summary>
    /// <param name="prompt">The prompt that you want Llama 2 to complete.</
param>
    /// <returns>The inference response from the model</returns>
    /// <remarks>
    /// The different model providers have individual request and response
formats.
    /// For the format, ranges, and default values for Meta Llama 2 Chat, refer
to:
```

```
/// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-meta.html
/// </remarks>
public static async Task<string> InvokeLlama2Async(string prompt)
{
    string llama2ModelId = "meta.llama2-13b-chat-v1";

    AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

    string payload = new JsonObject()
    {
        { "prompt", prompt },
        { "max_gen_len", 512 },
        { "temperature", 0.5 },
        { "top_p", 0.9 }
    }.ToJsonString();

    string generatedText = "";
    try
    {
        InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
        {
            ModelId = llama2ModelId,
            Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
            ContentType = "application/json",
            Accept = "application/json"
        });

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            return JsonNode.ParseAsync(response.Body)
                .Result?["generation"]?.GetValue<string>() ?? "";
        }
        else
        {
            Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
        }
    }
    catch (AmazonBedrockRuntimeException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

```
        return generatedText;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModel](#) 中的。

场景

创建用于与 Amazon Bedrock 基础模型进行交互的操场应用程序

以下代码示例演示如何创建操场，以通过不同模态与 Amazon Bedrock 基础模型交互。

AWS SDK for .NET

.NET Foundation Model (FM) Playground 是一个 .NET MAUI Blazor 示例应用程序，演示了如何通过 C# 代码使用 Amazon Bedrock。此示例演示了 .NET 和 C# 开发人员如何使用 Amazon Bedrock 来构建生成式人工智能赋能的应用程序。您可以使用以下四个操场测试 Amazon Bedrock 基础模型并与其交互：

- 文本操场。
- 聊天操场。
- 语音聊天操场。
- 图像操场。

该示例还列出并显示了您可以访问的基础模型及其特征。有关源代码和部署说明，请参阅中的项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock 运行时系统

CloudWatch 使用示例 AWS SDK for .NET

以下代码示例向您展示了如何使用 `with` 来执行操作和实现常见场景 CloudWatch。AWS SDK for .NET

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。


每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 CloudWatch

以下代码示例演示了如何开始使用 CloudWatch。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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($"Hello Amazon CloudWatch! Following are some metrics
available in the {metricNamespace} namespace:");
Console.WriteLine();
foreach (var metric in response.Metrics.Take(5))
{
    Console.WriteLine($"Metric: {metric.MetricName}");
    Console.WriteLine($"Namespace: {metric.Namespace}");
    Console.WriteLine($"Dimensions: {string.Join(", ",
metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
    Console.WriteLine();
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListMetrics](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建控制面板

以下代码示例显示了如何创建 Amazon CloudWatch 控制面板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutDashboard](#) 中的。

创建指标警报

以下代码示例演示如何创建或更新 Amazon CloudWatch 警报，并将其与指定指标、指标数学表达式、异常检测模型或指标见解查询相关联。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutMetricAlarm](#) 中的。

创建异常检测器

以下代码示例显示了如何创建 Amazon CloudWatch 异常检测器。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutAnomalyDetector](#) 中的。

删除警报

以下代码示例显示了如何删除 Amazon CloudWatch 警报。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteAlarms](#) 中的。

删除异常检测器

以下代码示例显示如何删除 Amazon CloudWatch 异常检测器。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteAnomalyDetector](#) 中的。

删除控制面板

以下代码示例显示了如何删除 Amazon CloudWatch 控制面板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```


- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [Delete Dashboards](#) 中的。

描述告警历史记录

以下代码示例显示了如何描述 Amazon CloudWatch 警报历史记录。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAlarmHistory](#) 中的。

描述警报

以下代码示例显示了如何描述 Amazon CloudWatch 警报。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAlarms](#) 中的。

为指标描述警报

以下代码示例显示了如何描述某个指标的 Amazon CloudWatch 警报。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAlarmsForMetric](#) 中的。

描述异常检测器

以下代码示例显示了如何描述 Amazon CloudWatch 异常检测器。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAnomalyDetectors](#) 中的。

禁用警报操作

以下代码示例显示了如何禁用 Amazon CloudWatch 警报操作。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DisableAlarmActions](#) 中的。

启用警报操作

以下代码示例显示了如何启用 Amazon CloudWatch 警报操作。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [EnableAlarmActions](#) 中的。

获取指标数据图片

以下代码示例显示了如何获取 Amazon CloudWatch 指标数据图片。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetMetricWidgetImage](#) 中的。

获取控制面板详细信息

以下代码示例显示了如何获取 Amazon CloudWatch 控制面板的详细信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetDashboard](#) 中的。

获取指标数据

以下代码示例显示了如何获取 Amazon CloudWatch 指标数据。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetMetricData](#) 中的。

获取指标统计数据

以下代码示例显示了如何获取 Amazon CloudWatch 指标统计数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetMetricStatistics](#) 中的。

列出控制面板

以下代码示例显示了如何列出 Amazon CloudWatch 控制面板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListDashboards](#) 中的。

列出指标

以下代码示例显示如何列出 Amazon CloudWatch 指标的元数据。要获取指标的数据，请使用 `GetMetricData` 或 `GetMetricStatistics` 操作。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListMetrics](#) 中的。

将数据放入指标

以下代码示例展示了如何将指标数据点发布到 Amazon CloudWatch。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutMetricData](#) 中的。

场景

开始使用 CloudWatch 指标、控制面板和告警

以下代码示例显示了操作流程：

- 列出 CloudWatch 命名空间和指标。
- 获取指标和预估账单的统计数据。
- 创建和更新控制面板。
- 创建数据并将数据添加到指标。
- 创建并触发告警，然后查看告警历史记录。
- 添加异常检测器
- 获取指标映像，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
public class CloudWatchScenario
{
    /*
     Before running this .NET code example, set up your development environment,
     including your credentials.

     To enable billing metrics and statistics for this example, make sure billing
     alerts are enabled for your account:
     https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
     monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

     This .NET example performs the following tasks:
     1. List and select a CloudWatch namespace.
     2. List and select a CloudWatch metric.
     3. Get statistics for a CloudWatch metric.
     4. Get estimated billing statistics for the last week.
     5. Create a new CloudWatch dashboard with two metrics.
     6. List current CloudWatch dashboards.
     7. Create a CloudWatch custom metric and add metric data.
     8. Add the custom metric to the dashboard.
     9. Create a CloudWatch alarm for the custom metric.
    10. Describe current CloudWatch alarms.
    11. Get recent data for the custom metric.
    12. Add data to the custom metric to trigger the alarm.
    13. Wait for an alarm state.
    14. Get history for the CloudWatch alarm.
    15. Add an anomaly detector.
    16. Describe current anomaly detectors.
    17. Get and display a metric image.
    18. Clean up resources.
    */

    private static ILogger logger = null!;
```



```
private static CloudWatchWrapper _cloudWatchWrapper = null!;
private static IConfiguration _configuration = null!;
private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
private static SingleMetricAnomalyDetector? anomalyDetector = null!;

static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
                .AddTransient<CloudWatchWrapper>()
        )
        .Build();

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

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<CloudWatchScenario>();

    _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
    Console.WriteLine(new string('-', 80));

    try
    {
        var selectedNamespace = await SelectNamespace();
        var selectedMetric = await SelectMetric(selectedNamespace);
        await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
        await GetAndDisplayEstimatedBilling();
        await CreateDashboardWithMetrics();
    }
}
```

```

        await ListDashboards();
        await CreateNewCustomMetric();
        await AddMetricToDashboard();
        await CreateMetricAlarm();
        await DescribeAlarms();
        await GetCustomMetricData();
        await AddMetricDataForAlarm();
        await CheckForMetricAlarm();
        await GetAlarmHistory();
        anomalyDetector = await AddAnomalyDetector();
        await DescribeAnomalyDetectors();
        await GetAndOpenMetricImage();
        await CleanupResources();
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
        await CleanupResources();
    }
}

/// <summary>
/// Select a namespace.
/// </summary>
/// <returns>The selected namespace.</returns>
private static async Task<string> SelectNamespace()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
    var metrics = await _cloudWatchWrapper.ListMetrics();
    // Get a distinct list of namespaces.
    var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
    for (int i = 0; i < namespaces.Count; i++)
    {
        Console.WriteLine($"{i + 1}. {namespaces[i]}");
    }

    var namespaceChoiceNumber = 0;
    while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
    {
        Console.WriteLine(

```

```
        "Select a namespace by entering a number from the preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out namespaceChoiceNumber);
    }

    var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

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

    return selectedNamespace;
}

/// <summary>
/// Select a metric from a namespace.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <returns>The metric name.</returns>
private static async Task<Metric> SelectMetric(string metricNamespace)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

    var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

    for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
    {
        var dimensionsWithValues = namespaceMetrics[i].Dimensions
            .Where(d => !string.Equals("None", d.Value));
        Console.WriteLine($"  \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($"{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
list:");
                "Select a metric statistic by entering a number from the preceding
            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($"\tDelete alarm {alarmName}? (y/n)"))
    {
        Console.WriteLine($"Cleaning up alarms.");
        var alarms = new List<string> { alarmName };
        await _cloudWatchWrapper.DeleteAlarms(alarms);
    }

    if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
        anomalyDetector != null)
    {
        Console.WriteLine($"Cleaning up anomaly detector.");

        await _cloudWatchWrapper.DeleteAnomalyDetector(
            anomalyDetector);
    }

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

/// <summary>
/// Get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null &&
        ynResponse.Equals("y",
            StringComparison.InvariantCultureIgnoreCase);

    return response;
}
}

```

场景用于 CloudWatch 操作的封装方法。

```

/// <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>
/// <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;
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [DeleteAlarms](#)
- [DeleteAnomalyDetector](#)
- [DeleteDashboards](#)
- [DescribeAlarmHistory](#)
- [DescribeAlarms](#)
- [DescribeAlarmsForMetric](#)

- [DescribeAnomalyDetectors](#)
- [GetMetricData](#)
- [GetMetricStatistics](#)
- [GetMetricWidgetImage](#)
- [ListMetrics](#)
- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)
- [PutMetricData](#)

CloudWatch 使用记录示例 AWS SDK for .NET

以下代码示例向您展示了如何使用 `with Logs` 来执行操作和实现常见场 CloudWatch 景。AWS SDK for .NET

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

将密钥与日志组相关联

以下代码示例显示如何将AWS KMS密钥与现有 CloudWatch 日志组关联。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}.");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AssociateKmsKey](#) 中的。

取消导出任务

以下代码示例显示如何取消现有的 CloudWatch 日志导出任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CancelExportTask](#) 中的。

创建日志组

以下代码示例显示了如何创建新的 CloudWatch 日志组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateLogGroup](#) 中的。

创建新的日志流

以下代码示例显示了如何创建新的 Lo CloudWatch gs 日志流。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateLogStream](#) 中的。

创建导出任务

以下代码示例显示了如何创建新的 CloudWatch 日志导出任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateExportTask](#) 中的。

删除日志组

以下代码示例显示如何删除现有的 CloudWatch 日志组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteLogGroup](#) 中的。

描述导出任务

以下代码示例显示了如何描述 CloudWatch 日志导出任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeExportTasks](#) 中的。

描述日志组

以下代码示例显示了如何描述 CloudWatch 日志组。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeLogGroups](#) 中的。

开始 Live Tail 会话

以下代码示例演示了如何为现有日志组/日志流启动 Live Tail 会话。

AWS SDK for .NET

包含所需的文件。

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

启动 Live Tail 会话

```
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;
}
```

您可以通过两种方式处理 Live Tail 会话中的事件：

```
/* 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");
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartLiveTail](#) 中的。

使用 AWS SDK for .NET 的 Amazon Cognito 身份提供商示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Cognito 身份提供者结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)
- [场景](#)

操作

确认用户

以下代码示例演示了如何确认 Amazon Cognito 用户。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ConfirmSignUp](#) 中的。

确认 MFA 设备可供跟踪

以下代码示例演示了如何确认 MFA 设备可通过 Amazon Cognito 进行跟踪。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ConfirmDevice](#) 中的。

获取令牌以将 MFA 应用程序与用户关联

以下代码示例演示了如何获取令牌，以将 MFA 应用程序与 Amazon Cognito 用户关联。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AssociateSoftwareToken](#) 中的。

获取有关用户的信息

以下代码示例演示了如何获取有关 Amazon Cognito 用户的信息。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AdminGetUser](#) 中的。

列出用户池

以下代码示例显示如何列出 Amazon Cognito 用户池。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListUserPools](#) 中的。

列出用户

以下代码示例演示了如何列出 Amazon Cognito 用户。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListUsers](#) 中的。

重新发送确认码

以下代码示例演示了如何重新发送 Amazon Cognito 确认码。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ResendConfirmationCode](#) 中的。

响应身份验证质询

以下代码示例演示了如何响应 Amazon Cognito 身份验证质询。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AdminRespondToAuthChallenge](#) 中的。

注册用户

以下代码示例演示了如何向 Amazon Cognito 注册用户。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SignUp](#) 中的。

开始身份验证

以下代码示例演示了如何通过 Amazon Cognito 开始身份验证。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InitiateAuth](#) 中的。

使用管理员凭证开始身份验证

以下代码示例演示了如何使用 Amazon Cognito 和管理员凭证开始身份验证。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AdminInitiateAuth](#) 中的。

向用户验证 MFA 应用程序

以下代码示例演示了如何向 Amazon Cognito 用户验证 MFA 应用程序。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [VerifySoftwareToken](#) 中的。

场景

向需要 MFA 的用户池注册用户

以下代码示例演示了操作流程：

- 使用用户名、密码和电子邮件地址注册和确认用户。
- 通过将 MFA 应用程序与用户关联来设置多重身份验证。
- 使用密码和 MFA 代码登录。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [AdminGetUser](#)

- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)
- [AssociateSoftwareToken](#)
- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

使用 AWS SDK for .NET 的 Amazon Comprehend 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Comprehend 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

检测文档中的实体

以下代码示例演示如何通过 Amazon Comprehend 检测文档中的实体。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");  
  }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectEntities](#) 中的。

检测文档中的关键短语

以下代码示例演示如何通过 Amazon Comprehend 检测文档中的关键短语。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectKeyPhrases](#) 中的。

检测文档中的个人身份信息

以下代码示例演示如何通过 Amazon Comprehend 检测文档中的个人身份信息 (PII)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
```

```
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally identifiable information within the supplied text.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();
        var text = @"Hello Paul Santos. The latest statement for your
                    credit card account 1111-0000-1111-0000 was
                    mailed to 123 Any Street, Seattle, WA 98109.";

        var request = new DetectPiiEntitiesRequest
        {
            Text = text,
            LanguageCode = "EN",
        };

        var response = await comprehendClient.DetectPiiEntitiesAsync(request);

        if (response.Entities.Count > 0)
        {
            foreach (var entity in response.Entities)
            {
                var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                Console.WriteLine($"{entity.Type}: {entityValue}");
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectPiiEntities](#) 中的。

检测文档的语法元素

以下代码示例演示如何通过 Amazon Comprehend 检测文档的语法元素。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectSyntax](#) 中的。

检测文档中的主要语言

以下代码示例演示如何通过 Amazon Comprehend 检测文档中的主导语言。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectDominantLanguage](#) 中的。

检测文档的情绪

以下代码示例演示如何通过 Amazon Comprehend 检测文档的情绪。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using System;
```

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

/// <summary>
/// This example shows how to detect the overall sentiment of the supplied
/// text using the Amazon Comprehend service.
/// </summary>
public static class DetectSentiment
{
    /// <summary>
    /// This method calls the DetectSentimentAsync method to analyze the
    /// supplied text and determine the overall sentiment.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

        // Call DetectKeyPhrases API
        Console.WriteLine("Calling DetectSentiment");
        var detectSentimentRequest = new DetectSentimentRequest()
        {
            Text = text,
            LanguageCode = "en",
        };
        var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
        Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
        Console.WriteLine("Done");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectSentiment](#) 中的。

开始主题建模任务

以下代码示例演示如何启动 Amazon Comprehend 主题建模任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartTopicsDetectionJob](#) 中的。

使用 AWS SDK for .NET 的 DynamoDB 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 DynamoDB 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 DynamoDB

以下代码示例演示了如何开始使用 DynamoDB。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTables](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建表

以下代码示例演示了如何创建 DynamoDB 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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.WriteLine("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.WriteLine(".");
}
while (status != "ACTIVE");
```

```
        return status == TableStatus.ACTIVE;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateTable](#) 中的。

删除表

以下代码示例演示了如何删除 DynamoDB 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTable](#) 中的。

删除表中项目

以下代码示例演示了如何删除 DynamoDB 表中的项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteItem](#) 中的。

获取一批项目

以下代码示例演示了如何获取一批 DynamoDB 项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
}
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [BatchGetItem](#) 中的。

获取表中项目

以下代码示例演示了如何从 DynamoDB 表中获取项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetItem](#) 中的。

获取表信息

以下代码示例演示了如何获取有关 DynamoDB 表的信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeTable](#) 中的。

列出表

以下代码示例演示了如何列出 DynamoDB 表。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTables](#) 中的。

向表中加入项目

以下代码示例演示了如何将项目放入 DynamoDB 表中。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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 information for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
added.</param>
    /// <returns>A Boolean value that indicates the results of adding the
item.</returns>
    public 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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutItem](#) 中的。

查询表

以下代码示例演示了如何查询 DynamoDB 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Query](#)。

运行 PartiQL 语句

以下代码示例演示了如何在 DynamoDB 表上运行 PartiQL 语句。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用 INSERT 语句添加项目。

```
/// <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;
    }

```

使用 SELECT 语句获取项目。

```

    /// <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;
}
```

使用 SELECT 语句获取项目列表。

```
/// <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;
}
```

使用 UPDATE 语句更新项目。

```
/// <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;
}
```

使用 DELETE 语句删除单个电影。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ExecuteStatement](#) 中的。

运行批量 PartiQL 语句

以下代码示例演示了如何在 DynamoDB 表中运行批量 PartiQL 语句。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用批量 INSERT 语句添加项目。

```
/// <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!;
    }
}
```

使用批量 SELECT 语句获取项目。

```
/// <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;
    }
}
}

```

使用批量 UPDATE 语句更新项目。

```

/// <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;
}

```

使用批量 DELETE 语句删除项目。

```

/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [BatchExecuteStatement](#) 中的。

扫描表

以下代码示例演示了如何扫描 DynamoDB 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Scan](#)。

更新表中项目

以下代码示例演示了如何更新 DynamoDB 表中的项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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;  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateItem](#) 中的。

写入一批项目

以下代码示例演示了如何写入一批 DynamoDB 项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

将一批项目写入影片表。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [BatchWriteItem](#) 中的。

场景

开始使用表、项目和查询

以下代码示例显示了操作流程：

- 创建可保存电影数据的表。
- 在表中加入单一电影，获取并更新此电影。
- 向 JSON 示例文件的表中写入电影数据。
- 查询在给定年份发行的电影。
- 扫描在年份范围内发行的电影。
- 删除表中的电影后再删除表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
// 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($"
Table: {tableName} successfully created.");
        }
        else
        {
            Console.WriteLine($"
Could 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();
}
}
```

创建一个包含影片数据的表。

```
    /// <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.WriteLine("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.WriteLine(".");
    }
    while (status != "ACTIVE");

    return status == TableStatus.ACTIVE;
}
```

将单个影片添加到表中。

```
/// <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 information for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
added.</param>
    /// <returns>A Boolean value that indicates the results of adding the
item.</returns>
    public 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;
    }

```

更新表中的单个项目。

```

    /// <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;
}
```

从影片表中检索单个项目。

```

    /// <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;
    }

```

将一批项目写入影片表。

```

    /// <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;
    }
```

从表中删除单个项目。

```
    /// <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;
    }
```

查询表中是否有特定年份发行的影片。

```
/// <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;
}

```

扫描此表以查找几年内发行的影片。

```

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;
}
```

删除影片表。

```
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;
    }
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的以下主题。
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)


- [DescribeTable](#)
- [GetItem](#)
- [PutItem](#)
- [查询](#)
- [扫描](#)
- [UpdateItem](#)

使用批量 PartiQL 语句查询表

以下代码示例演示了操作流程：

- 通过运行多个 SELECT 语句来获取一批项目。
- 通过运行多个 INSERT 语句来添加一批项目。
- 通过运行多个 UPDATE 语句来更新一批项目。
- 通过运行多个 DELETE 语句来删除一批项目。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
// 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.WriteLine(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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [BatchExecuteStatement](#) 中的。

使用 PartiQL 来查询表

以下代码示例演示了操作流程：

- 通过运行 SELECT 语句来获取项目。
- 通过运行 INSERT 语句来添加项目。
- 通过运行 UPDATE 语句来更新项目。
- 通过运行 DELETE 语句来删除项目。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ExecuteStatement](#) 中的。

使用文档模型

以下代码示例显示如何使用 DynamoDB 的文档模型以及 AWS SDK 来执行创建、读取、更新和删除 (CRUD) 及批处理操作。

有关更多信息，请参阅[文档模型](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用文档模型执行 CRUD 操作。

```
/// <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.ExpressionAttributeValue[":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);
    }
}
```

使用文档模型执行批量写入操作。

```
/// <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();
}
}
```

使用文档模型扫描表。

```
/// <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}");
    }
}
}
```

使用文档模型查询和扫描表。

```
/// <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}");
}
}
```

使用高级对象持久化模型

以下代码示例显示如何使用 DynamoDB 的对象持久化模型和 AWS SDK 来执行创建、读取、更新和删除 (CRUD) 及批处理操作。

有关更多信息，请参阅[对象持久化模型](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用高级对象持久化模型执行 CRUD 操作。

```
/// <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");
    }
}
}

```

使用高级对象持久化模型执行批量写入操作。

```

/// <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);
    }
}
```

使用高级对象持久化模型将任意数据映射为表。

```
/// <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);
}
}
```

使用高级对象持久化模型查询和扫描表。

```
/// <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}");
}
}
```

使用 AWS SDK for .NET 的 Amazon EC2 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon EC2 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon EC2

以下代码示例演示了如何开始使用 Amazon EC2。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
});
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeSecurityGroups](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

分配弹性 IP 地址

以下代码示例演示了如何为 Amazon EC2 分配弹性 IP 地址。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AllocateAddress](#) 中的。

关联弹性 IP 地址和实例

以下代码示例演示了如何将弹性 IP 地址与 Amazon EC2 实例关联。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AssociateAddress](#) 中的。

创建启动模板

以下代码示例演示了如何创建 Amazon EC2 启动模板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateLaunchTemplate](#) 中的。

创建安全组

以下代码示例演示了如何创建 Amazon EC2 安全组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateSecurityGroup](#) 中的。

创建安全密钥对

以下代码示例演示了如何为 Amazon EC2 创建安全密钥对。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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 SaveKeyValuePair(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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateKeyValuePair](#) 中的。

创建并运行实例

以下代码示例演示了如何创建并运行 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RunInstances](#) 中的。

删除启动模板

以下代码示例演示了如何删除 Amazon EC2 启动模板。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteLaunchTemplate](#) 中的。

删除安全组

以下代码示例演示了如何删除 Amazon EC2 安全组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteSecurityGroup](#) 中的。

删除安全密钥对

以下代码示例演示如何删除 Amazon EC2 安全密钥对。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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);
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteKeyPair](#) 中的。

描述可用区

以下代码示例演示如何描述 Amazon EC2 可用区。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAvailabilityZones](#) 中的。

描述实例

以下代码示例演示如何描述 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}");
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeInstances](#) 中的。

取消弹性 IP 地址与实例的关联

以下代码示例演示了如何取消弹性 IP 地址与 Amazon EC2 实例的关联。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DisassociateAddress](#) 中的。

获取有关安全组的数据

以下代码示例演示了如何获取有关 Amazon EC2 安全组的数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"{range.CidrIp}
"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.WriteLine($"{range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{id.Id} "));

        Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");
    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.WriteLine($"{range.CidrIp}
"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.WriteLine($"{range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{id.Id} "));

        Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");
    });
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeSecurityGroups](#) 中的。

获取有关实例类型的数据

以下代码示例演示了如何获取有关 Amazon EC2 实例类型的数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeInstanceTypes](#) 中的。

获取有关与实例关联的实例配置文件的数据

以下代码示例演示了如何获取有关与 Amazon EC2 实例关联的实例配置文件的数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeIamInstanceProfileAssociations](#) 中的。

获取默认 VPC

以下代码示例演示了如何获取当前账户的默认 VPC。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeVpcs](#) 中的。

获取 VPC 的默认子网

以下代码示例演示了如何获取 VPC 的默认子网。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeSubnets](#) 中的。

列出安全密钥对

以下代码示例演示了如何列出 Amazon EC2 安全密钥对。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeKeyPairs](#) 中的。

重启实例

以下代码示例演示了如何重启 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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.");
    }
}
```

替换实例的配置文件、重启并重新启动 Web 服务器。

```
/// <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}");
    }

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RebootInstances](#) 中的。

释放弹性 IP 地址

以下代码示例演示了如何释放弹性 IP 地址。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ReleaseAddress](#) 中的。

替换与实例关联的实例配置文件

以下代码示例演示了如何替换与 Amazon EC2 实例关联的实例配置文件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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}");
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ReplaceInstanceProfileAssociation](#) 中的。

为安全组设置进站规则

以下代码示例演示了如何为 Amazon EC2 安全组设置进站规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AuthorizeSecurityGroupIngress](#) 中的。

启动实例

以下代码示例演示如何启动 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}.");
    });
}
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartInstances](#) 中的。

停止实例

以下代码示例演示如何停止 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

/// <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}.");
    });
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StopInstances](#) 中的。

终止实例

以下代码示例演示如何终止 Amazon EC2 实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [TerminateInstances](#) 中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud (Amazon EC2) 实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 AWS Systems Manager 参数控制 Web 服务器对请求和运行状况检查的响应。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

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

    ServicesSetup(host);
    ResourcesSetup();

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

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

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

        await DestroyResources(true);

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

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

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

```
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

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

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

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

        // Create and populate the DynamoDB table.
```

```
var databaseTableName = _configuration["databaseName"];
var recommendationsPath = Path.Join(_configuration["resourcePath"],
    "recommendations_objects.json");
Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
await _recommendations.CreateDatabaseWithName(databaseTableName);
await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
Console.WriteLine(new string('-', 80));

// Create the EC2 Launch Template.

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

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

Console.WriteLine(
    "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
    + "Availability Zone.\n");
var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
```

```
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

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

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

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

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

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

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



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

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

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

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

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

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

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                    "Do you want to add a rule to the security group to allow
                    inbound SSH traffic for debugging from your computer's IP address?"))
                {
```

```

        {
            await
            _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
            ipString);
        }
    }
    loadBalancerAccess = await
    _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

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

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

```

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

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

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

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

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

Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
```

```
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

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

```
        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

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

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

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

        if (interactive)
            await DemoActionChoices();

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

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"Even while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
```

```
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

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

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

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

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

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

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

```

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
            _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
            _recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

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

```

创建一个包含自动扩缩和 Amazon EC2 操作的类。

```

/// <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 = "{" +
        "\n\"Version\": \"2012-10-17\", " +
        "\n\"Statement\": [{" +
            "\n    \"Effect\": \"Allow\", " +
            "\n    \"Principal\": { " +
            "\n        \"Service\": [ " +
            "\n            \"ec2.amazonaws.com\" " +
            "\n        ] " +

```

```
        "}," +
        "\"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 = deleteKeyName });
        File.Delete($"{deleteKeyName}.pem");
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"Key pair {deleteKeyName} not found.");
    }
}

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

```

创建一个包含弹性负载均衡操作的类。

```

/// <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);
    }
}
}
```

创建一个使用 DynamoDB 模拟推荐服务的类。

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



```
public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

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

```
        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

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

var request = new DescribeTableRequest
{
    TableName = tableName
};

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

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

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

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

```
        return false;
    }
}

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

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

    await batchWrite.ExecuteAsync();
}

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

```
}  
}
```

创建一个包含 Systems Manager 操作的类。

```
/// <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 });
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [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](#)

开始使用实例

以下代码示例演示了操作流程：

- 创建密钥对和安全组。
- 选择 Amazon Machine Image (AMI) 和兼容的实例类型，然后创建实例。
- 停止实例，然后再重启。
- 将弹性 IP 地址与您的实例相关联。
- 使用 SSH 连接到您的实例，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行场景。

```
/// <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.WriteLine("Let's create an RSA key pair that you can 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.WriteLine("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($"{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($"{i++}\t{instanceType.InstanceType}");
    });

    do
    {
        Console.WriteLine("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.WriteLine("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("\\n\\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("\\n\\nLet's see what changed.");

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\\n\\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.WriteLine("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();
}
}

```

定义一个包装 EC2 操作的类。

```

/// <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\n\"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.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

        Console.WriteLine($"  \n\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

        Console.WriteLine($"  \n\tTo Port: {permission.ToPort}");
    });
    Console.WriteLine("Egress permissions:");
    securityGroup.IpPermissionsEgress.ForEach(permission =>
    {
        Console.WriteLine($"  \tFromPort: {permission.FromPort}");
        Console.WriteLine($"  \tIpProtocol: {permission.IpProtocol}");

        Console.WriteLine($"  \tIpv4Ranges: ");
        permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

        Console.WriteLine($"  \n\tIpv6Ranges:");
        permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });
    });
}
```

```
        Console.WriteLine($"\\n\\tPrefixListIds: ");
        permission.PrefixListIds.ForEach(id => Console.WriteLine($"{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.WriteLine(". ");
    } while (!hasState);

    return hasState;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [AllocateAddress](#)
- [AssociateAddress](#)
- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

使用 AWS SDK for .NET 的 Amazon ECS 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon ECS 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用 Amazon ECS

以下代码示例演示如何开始使用 Amazon ECS。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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($"  \tCluster Name: {arn.Split("/").Last()}");
            Console.WriteLine();
        }
    }
}
```



```
}  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListClusters](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

列出集群

以下代码示例演示如何列出 Amazon ECS 集群。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListClusters](#) 中的。

列出集群中的服务

以下代码示例演示如何列出集群中的 Amazon ECS 服务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
///  
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListServices](#) 中的。

列出集群中的任务

以下代码示例演示如何列出集群中的 Amazon ECS 任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTasks](#) 中的。

场景

获取集群、服务和任务的 ARN 信息

以下代码示例演示了操作流程：

- 获取所有集群的列表。
- 获取集群的服务。
- 获取集群的任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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));
}
}
```

场景调用以管理 Amazon ECS 操作的包装程序方法。

```
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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [ListClusters](#)
- [ListServices](#)
- [ListTasks](#)

使用 AWS SDK for .NET 的 Elastic Load Balancing 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Elastic Load Balancing 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)
- [场景](#)

操作

为负载均衡器创建侦听器

以下代码示例演示如何创建一个将请求从 ELB 负载均衡器转发到目标组的侦听器。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateListener](#) 中的。

创建目标组

以下代码示例演示如何创建 ELB 目标组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateTargetGroup](#) 中的。

创建应用程序负载均衡器

以下代码示例演示如何创建 ELB 应用程序负载均衡器。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateLoadBalancer](#) 中的。

删除负载均衡器

以下代码示例演示如何删除 ELB 负载均衡器。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteLoadBalancer](#) 中的。

删除目标组

以下代码示例演示如何删除 ELB 目标组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
///  
/// <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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTargetGroup](#) 中的。

获取负载均衡器的端点

以下代码示例演示如何获取 ELB 负载均衡器的端点。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeLoadBalancers](#) 中的。

获取目标组的运行状况

以下代码示例演示如何获取 ELB 目标组中实例的运行状况。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeTargetHealth](#) 中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud (Amazon EC2) 实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 AWS Systems Manager 参数控制 Web 服务器对请求和运行状况检查的响应。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();
}
```

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

ServicesSetup(host);
ResourcesSetup();

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

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

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

    await DestroyResources(true);

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

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

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

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

/// <summary>
/// Deploy necessary resources for the scenario.
/// </summary>
/// <param name="interactive">True to run as interactive.</param>
/// <returns>True if successful.</returns>
public static async Task<bool> Deploy(bool interactive)
```



```
{
    var protocol = "HTTP";
    var port = 80;
    var sshPort = 22;

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

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

    // Create and populate the DynamoDB table.
    var databaseTableName = _configuration["databaseName"];
    var recommendationsPath = Path.Join(_configuration["resourcePath"],
        "recommendations_objects.json");
    Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
    await _recommendations.CreateDatabaseWithName(databaseTableName);
    await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
    Console.WriteLine(new string('-', 80));

    // Create the EC2 Launch Template.
```

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

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

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

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

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

```
        Console.ReadLine();

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

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

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

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

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

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

            var defaultSecurityGroup = await
                _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
```

```
        var portIsOpen =
        _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
        var sshPortIsOpen =
        _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
        ipString);

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

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

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

        loadBalancerAccess = await
        _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
    }

    if (loadBalancerAccess)
    {
        Console.WriteLine("Your load balancer is ready. You can access it by
        browsing to:");
    }
}
```

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

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

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

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

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

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

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

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

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

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
    );
```

```
await _autoScalerWrapper.CreateInstanceProfileWithName(
    _autoScalerWrapper.BadCredsPolicyName,
    _autoScalerWrapper.BadCredsRoleName,
    _autoScalerWrapper.BadCredsProfileName,
    ssmOnlyPolicy,
    new List<string> { "AmazonSSMManagedInstanceCore" }
);
var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
var badInstanceId = instances.First();
var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
Console.WriteLine(
    $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
    "bad credentials...\n"
);
await _autoScalerWrapper.ReplaceInstanceProfile(
    badInstanceId,
    _autoScalerWrapper.BadCredsProfileName,
    instanceProfile.AssociationId
);
Console.WriteLine(
    "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
    "depending on which instance is selected by the load balancer.\n"
);
if (interactive)
    await DemoActionChoices();

Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

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

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

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

        if (interactive)
            await DemoActionChoices();

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

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

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

        if (interactive)
            await DemoActionChoices();

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

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



```

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

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

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

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

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
            _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,

```

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

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

创建一个包含自动扩缩和 Amazon EC2 操作的类。

```
/// <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 }  
            });  
    }  
}
```

创建一个包含弹性负载均衡操作的类。

```
/// <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.");
        }
    }
}

```

```

    }
    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);
    }
}
}
}
}

```

创建一个使用 DynamoDB 模拟推荐服务的类。

```

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

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }
}

```

```
/// <summary>
/// Create the DynamoDb table with a specified name.
/// </summary>
/// <param name="tableName">The name for the table.</param>
/// <returns>True when ready.</returns>
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.WriteLine($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition()
                {
                    AttributeName = "MediaType",
                    AttributeType = ScalarAttributeType.S
                },
                new AttributeDefinition()
                {
                    AttributeName = "ItemId",
                    AttributeType = ScalarAttributeType.N
                }
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement()
                {
                    AttributeName = "MediaType",
                    KeyType = KeyType.HASH
                },
                new KeySchemaElement()
                {
                    AttributeName = "ItemId",
                    KeyType = KeyType.RANGE
                }
            },
            ProvisionedThroughput = new ProvisionedThroughput()
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5
            }
        }
    }
}
```

```
        }
    };
    await _amazonDynamoDb.CreateTableAsync(createRequest);

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

    var request = new DescribeTableRequest
    {
        TableName = tableName
    };

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

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

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

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

/// <summary>
/// Populate the database table with data from a specified path.
/// </summary>
/// <param name="databaseTableName">The name of the table.</param>
/// <param name="recommendationsPath">The path of the recommendations data.</
param>
/// <returns>Async task.</returns>
public async Task PopulateDatabase(string databaseTableName, string
recommendationsPath)
{
```

```

var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
var records =
    JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

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

await batchWrite.ExecuteAsync();
}

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

```

创建一个包含 Systems Manager 操作的类。

```

/// <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 });
}
}
```

• 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)

- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

EventBridge 使用示例 AWS SDK for .NET

以下代码示例向您展示了如何使用with来执行操作和实现常见场景 EventBridge。AWS SDK for .NET

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 EventBridge

以下代码示例演示了如何开始使用 EventBridge。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListEventBuses](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

添加目标

以下代码示例显示了如何向 Amazon EventBridge 事件添加目标。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

添加 Amazon SNS 主题，作为规则的目标。

```
/// <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;
    }

```

将输入转换器添加到规则的目标。

```

/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutTargets](#) 中的。

创建规则

以下代码示例显示了如何创建 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建规则，将对象添加到 Amazon Simple Storage Service 桶时触发。

```
/// <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;
    }

```

创建使用自定义模式的规则。

```

    /// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutRule](#) 中的。

删除规则

以下代码示例显示了如何删除 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

按名称删除规则。

```
/// <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;  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteRule](#) 中的。

描述规则

以下代码示例显示了如何描述 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用规则描述获取规则的状态。

```
/// <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;  
}
```


- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeRule](#) 中的。

禁用规则

以下代码示例显示了如何禁用 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

按规则名称禁用规则。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DisableRule](#) 中的。

启用规则

以下代码示例显示了如何启用 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

按规则名称启用规则。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [EnableRule](#) 中的。

列出目标的规则名称

以下代码示例说明如何列出目标的 Amazon EventBridge 规则名称。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出使用此目标的所有规则名称。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListRuleNamesByTarget](#) 中的。

列出规则

以下代码示例显示了如何列出 Amazon EventBridge 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出某事件总线的所有规则。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListRules](#) 中的。

列出规则的目标

以下代码示例显示了如何列出规则的 Amazon EventBridge 目标。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用规则名称列出规则的所有目标。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTargetsByRule](#) 中的。

从规则中移除目标

以下代码示例显示如何从规则中移除 Amazon EventBridge 目标。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用规则名称删除该规则的所有目标。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RemoveTargets](#) 中的。

发送事件

以下代码示例显示了如何发送 Amazon EventBridge 事件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

发送与规则的自定义模式相匹配的事件。

```
/// <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;  
    }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutEvents](#) 中的。

场景

开始使用规则和目标

以下代码示例演示了操作流程：

- 创建规则并为其添加目标。
- 启用和禁用规则。
- 列出并更新规则和目标。
- 发送事件，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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;
}
}
```

创建一个封装 EventBridge 操作的类。

```
/// <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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [DeleteRule](#)
- [DescribeRule](#)
- [DisableRule](#)
- [EnableRule](#)
- [ListRuleNamesByTarget](#)
- [ListRules](#)
- [ListTargetsByRule](#)
- [PutEvents](#)
- [PutRule](#)
- [PutTargets](#)

使用 AWS SDK for .NET 的 AWS Glue 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 AWS Glue 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 AWS Glue

以下代码示例演示了如何开始使用 AWS Glue。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListJobs](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建爬网程序

以下代码示例演示了如何创建 AWS Glue 爬网程序。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateCrawler](#) 中的。

创建任务定义

以下代码示例演示如何创建 AWS Glue 任务定义。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateJob](#) 中的。

删除爬网程序

以下代码示例演示了如何删除 AWS Glue 爬网程序。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteCrawler](#) 中的。

从 Data Catalog 中删除数据库

以下代码示例演示了如何从 AWS Glue Data Catalog 中删除数据库。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteDatabase](#) 中的。

删除任务定义

以下代码示例演示如何删除 AWS Glue 任务定义和所有相关运行。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteJob](#) 中的。

从数据库中删除表

以下代码示例演示了如何从 AWS Glue Data Catalog 数据库中删除表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```


- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTable](#) 中的。

获取爬网程序

以下代码示例演示了如何获取 AWS Glue 爬网程序。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetCrawler](#) 中的。

从 Data Catalog 获取数据库

以下代码示例演示了如何从 AWS Glue Data Catalog 中获取数据库。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetDatabase](#) 中的。

让任务运行

以下代码示例演示如何让 AWS Glue 任务运行。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetJobRun](#) 中的。

获取任务运行情况

以下代码示例演示如何获取 AWS Glue 任务运行情况。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetJobRuns](#) 中的。

从数据库获取表

以下代码示例演示了如何在 AWS Glue Data Catalog 中从数据库获取表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetTables](#) 中的。

列出任务定义。

以下代码示例演示如何列出 AWS Glue 任务定义。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListJobs](#) 中的。

启动爬网程序

以下代码示例演示了如何启动 AWS Glue 爬网程序。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartCrawler](#) 中的。

启动作业运行

以下代码示例演示如何启动 AWS Glue 任务运行。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartJobRun](#) 中的。

场景

开始使用爬网程序和任务

以下代码示例演示了操作流程：

- 创建爬网程序，爬取公有 Amazon S3 存储桶并生成包含 CSV 格式的元数据的数据库。
- 在 AWS Glue Data Catalog 中列出有关数据库和表的信息。
- 创建任务，从 S3 存储桶提取 CSV 数据，转换数据，然后将 JSON 格式的输出加载到另一个 S3 存储桶中。
- 列出有关任务运行的信息，查看转换后的数据，并清理资源。

有关更多信息，请参阅 [教程：AWS Glue Studio 入门](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建一个包含场景中使用的 AWS Glue 函数的类。

```
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;
}
}
```

创建运行场景的类。


```
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.WriteLine("\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);
}
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的以下主题。

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)

- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

使用 AWS SDK for .NET 的 IAM 示例

以下代码示例演示了如何将AWS SDK for .NET 与 IAM 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 IAM

以下代码示例演示了如何开始使用 IAM。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
});
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListPolicies](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

将用户添加到组

以下代码示例显示如何将用户添加到 IAM 组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AddUserToGroup](#) 中的。

将策略附加到角色

以下代码示例演示了如何将 IAM 策略添加到角色。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AttachRolePolicy](#) 中的。

将内联策略附加到角色

以下代码示例演示了如何将内联策略附加到 IAM 角色。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutRolePolicy](#) 中的。

创建组

以下代码示例演示了如何创建 IAM 组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateGroup](#) 中的。

创建策略

以下代码示例演示如何创建 IAM 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreatePolicy](#) 中的。

创建角色

以下代码示例演示了如何创建 IAM 角色。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateRole](#) 中的。

创建服务相关角色

以下代码示例演示了如何创建 IAM 服务相关角色。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateServiceLinkedRole](#) 中的。


创建用户

以下代码示例演示了如何创建 IAM 用户。

 Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

 Note


还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateUser](#) 中的。

创建访问密钥

以下代码示例演示了如何创建 IAM 访问密钥。

 Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateAccessKey](#) 中的。

为组创建内联策略

以下代码示例演示了如何为组创建内联 IAM policy。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutGroupPolicy](#) 中的。

创建实例配置文件

以下代码示例演示了如何创建 IAM 实例配置文件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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 = "{" +
        "\nVersion\": \"2012-10-17\", \" +
        "\nStatement\": [{" +
            "\nEffect\": \"Allow\", \" +
            "\nPrincipal\": {\" +
            "\nService\": [\" +
                "\nec2.amazonaws.com\"\" +
            "]" +
            "}, \" +
        "\nAction\": \"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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateInstanceProfile](#) 中的。

删除组

以下代码示例演示了如何删除 IAM 组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteGroup](#) 中的。

删除组策略

以下代码示例显示如何删除 IAM 组策略。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteGroupPolicy](#) 中的。

删除策略

以下代码示例演示如何删除 IAM 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeletePolicy](#) 中的。

删除角色

以下代码示例演示了如何删除 IAM 角色。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteRole](#) 中的。

删除角色策略

以下代码示例演示了如何删除 IAM 角色策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteRolePolicy](#) 中的。

删除用户

以下代码示例演示了如何删除 IAM 用户。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteUser](#) 中的。

删除访问密钥

以下代码示例演示了如何删除 IAM 访问密钥。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteAccessKey](#) 中的。

删除用户的内联策略

以下代码示例演示如何删除用户的内联 IAM 策略。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteUserPolicy](#) 中的。

删除实例配置文件

以下代码示例演示了如何删除 IAM 实例配置文件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteInstanceProfile](#) 中的。

从角色分离策略

以下代码示例演示了如何从角色分离 IAM 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetachRolePolicy](#) 中的。

获取策略

以下代码示例演示了如何获取 IAM 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetPolicy](#) 中的。

获取角色

以下代码示例演示了如何获取 IAM 角色。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetRole](#) 中的。

获取用户

以下代码示例显示如何获取 IAM 用户。

⚠ Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

i Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetUser](#) 中的。

获取账户密码策略

以下代码示例演示了如何获取 IAM 账户密码策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetAccountPasswordPolicy](#) 中的。

列出 SAML 提供商

以下代码示例演示了如何列出 IAM 的 SAML 提供商。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。查找完整示例，了解如何在 [AWS 代码示例存储库](#) 中进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [ListSAMLProviders](#)。

列出组

以下代码示例演示了如何列出 IAM 组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListGroups](#) 中的。

列出角色的内联策略

以下代码示例演示了如何列出 IAM 角色的内联策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListRolePolicies](#) 中的。

列出策略

以下代码示例演示了如何列出 IAM 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListPolicies](#) 中的。

列出附加到角色的策略

以下代码示例演示了如何列出附加到 IAM 角色的策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListAttachedRolePolicies](#) 中的。

列出角色

以下代码示例演示了如何列出 IAM 角色。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListRoles](#) 中的。

列出用户

以下代码示例演示了如何列出 IAM 用户。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListUsers](#) 中的。

从组中删除用户

以下代码示例显示如何从 IAM 组中删除用户。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RemoveUserFromGroup](#) 中的。

场景

构建和管理弹性服务

以下代码示例演示了如何创建可返回书籍、电影和歌曲推荐的负载均衡的 Web 服务。该示例演示服务如何响应故障，以及如何重组服务以提高故障发生时的弹性。

- 使用 Amazon EC2 Auto Scaling 组根据启动模板创建 Amazon Elastic Compute Cloud (Amazon EC2) 实例，并将实例数量保持在指定范围内。
- 使用弹性负载均衡处理和分发 HTTP 请求。
- 监控自动扩缩组中实例的运行状况，并仅将请求转发到运行状况良好的实例。
- 在每个 EC2 实例上运行 Python Web 服务器以处理 HTTP 请求。Web 服务器以建议和运行状况检查作为响应。
- 使用 Amazon DynamoDB 表模拟推荐服务。
- 通过更新 AWS Systems Manager 参数控制 Web 服务器对请求和运行状况检查的响应。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

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

    ServicesSetup(host);
    ResourcesSetup();

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

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

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

        await DestroyResources(true);

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

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

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

```
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

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

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

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

        // Create and populate the DynamoDB table.
```

```
var databaseTableName = _configuration["databaseName"];
var recommendationsPath = Path.Join(_configuration["resourcePath"],
    "recommendations_objects.json");
Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
await _recommendations.CreateDatabaseWithName(databaseTableName);
await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
Console.WriteLine(new string('-', 80));

// Create the EC2 Launch Template.

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

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

Console.WriteLine(
    "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
    + "Availability Zone.\n");
var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
```

```
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

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

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

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

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

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

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

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

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

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

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

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

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

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                    "Do you want to add a rule to the security group to allow
                    inbound SSH traffic for debugging from your computer's IP address?"))
                {
```

```

        {
            await
            _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
            ipString);
        }
    }
    loadBalancerAccess = await
    _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
}

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

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

```



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

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

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

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

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

Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
```

```
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

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

```
        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

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

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

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

        if (interactive)
            await DemoActionChoices();

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

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"Even while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
```

```
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

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

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

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

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

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

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

```

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
            _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalancerName);
            await
            _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroupName);
            await
            _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName);
            await
            _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
            _autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
            _recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

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

```

创建一个包含自动扩缩和 Amazon EC2 操作的类。

```

/// <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 = "{" +
        "\n\"Version\": \"2012-10-17\", " +
        "\n\"Statement\": [{" +
            "\n    \"Effect\": \"Allow\", " +
            "\n    \"Principal\": {" +
            "\n        \"Service\": [" +
            "\n            \"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 = deleteKeyName });
    File.Delete($"{deleteKeyName}.pem");
}
catch (FileNotFoundException)
{
    Console.WriteLine($"Key pair {deleteKeyName} not found.");
}
}

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

```

创建一个包含弹性负载均衡操作的类。

```

/// <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);
        }
    }
}
}

```

创建一个使用 DynamoDB 模拟推荐服务的类。

```

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

```

```
public string TableName => _tableName;

/// <summary>
/// Constructor for the Recommendations service.
/// </summary>
/// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
/// <param name="configuration">The injected configuration.</param>
public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
configuration)
{
    _amazonDynamoDb = amazonDynamoDb;
    _context = new DynamoDBContext(_amazonDynamoDb);
    _tableName = configuration["databaseName"]!;
}

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

```
        {
            AttributeName = "MediaType",
            KeyType = KeyType.HASH
        },
        new KeySchemaElement()
        {
            AttributeName = "ItemId",
            KeyType = KeyType.RANGE
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput()
    {
        ReadCapacityUnits = 5,
        WriteCapacityUnits = 5
    }
};
await _amazonDynamoDb.CreateTableAsync(createRequest);

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

var request = new DescribeTableRequest
{
    TableName = tableName
};

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

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

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

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



```
        return false;
    }
}

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

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

    await batchWrite.ExecuteAsync();
}

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

```
}  
}
```

创建一个包含 Systems Manager 操作的类。

```
/// <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 });
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)

- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

创建组并添加用户

以下代码示例演示了操作流程：

- 创建组并向其授予完整的 Amazon S3 访问权限。
- 创建无权访问 Amazon S3 的新用户。
- 将用户添加到组并显示他们现在拥有的 Amazon S3 权限，然后清除资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.WriteLine("\nPress <Enter> to continue. ");
    _ = Console.ReadLine();
    Console.WriteLine();
}

/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter">The string to be centered.</param>
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title, and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);
}
```

```
// Wait for the requested number of seconds.
for (int i = numSeconds; i > 0; i--)
{
    System.Threading.Thread.Sleep(1000);
    Console.WriteLine($"{i}...");
}

PressEnter();
}
}
```

• 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [AddUserToGroup](#)
- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreateGroup](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeleteGroup](#)
- [DeleteGroupPolicy](#)
- [DeleteUser](#)
- [PutGroupPolicy](#)
- [RemoveUserFromGroup](#)

创建用户并代入角色

以下代码示例演示了如何创建用户并代入角色。

Warning

为了避免安全风险，在开发专用软件或处理真实数据时，请勿使用 IAM 用户进行身份验证，而是使用与身份提供商的联合身份验证，例如 [AWS IAM Identity Center](#)。

- 创建没有权限的用户。
- 创建授予列出账户的 Amazon S3 存储桶的权限的角色
- 添加策略以允许用户代入该角色。
- 代入角色并使用临时凭证列出 S3 存储桶，然后清除资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.WriteLine("\nPress <Enter> to continue. ");
    _ = Console.ReadLine();
    Console.WriteLine();
}

/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter">The string to be centered.</param>
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title, and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);

    // Wait for the requested number of seconds.
}
```

```
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.WriteLine($"{i}...");
        }

        PressEnter();
    }
}
```

• 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [AttachRolePolicy](#)
- [CreateAccessKey](#)
- [CreatePolicy](#)
- [CreateRole](#)
- [CreateUser](#)
- [DeleteAccessKey](#)
- [DeletePolicy](#)
- [DeleteRole](#)
- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

使用 AWS SDK for .NET 的 Amazon Keyspaces 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Keyspaces 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon Keyspaces

以下代码示例演示如何开始使用 Amazon Keyspaces。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListKeyspaces](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建键空间

以下代码示例演示如何创建 Amazon Keyspaces 键空间。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateKeyspace](#) 中的。

创建表

以下代码示例演示如何创建 Amazon Keyspaces 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
    }
```



```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateTable](#) 中的。

删除键空间

以下代码示例演示如何删除 Amazon Keyspaces 键空间。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteKeyspace](#) 中的。

删除表

以下代码示例演示如何删除 Amazon Keyspaces 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTable](#) 中的。

获取有关键空间的数据

以下代码示例演示如何获取 Amazon Keyspaces 键空间的相关数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetKeyspace](#) 中的。

获取有关表的数据

以下代码示例演示如何获取 Amazon Keyspaces 表的相关数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetTable](#) 中的。

列出键空间

以下代码示例演示如何列出 Amazon Keyspaces 键空间。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListKeyspaces](#) 中的。

列出键空间中的表

以下代码示例演示如何列出键空间中的 Amazon Keyspaces 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTables](#) 中的。

将表还原到某个时间点

以下代码示例演示如何将 Amazon Keyspaces 表还原到某个时间点。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RestoreTable](#) 中的。

更新表

以下代码示例演示如何更新 Amazon Keyspaces 表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateTable](#) 中的。

场景

开始使用键空间和表

以下代码示例演示了操作流程：

- 创建键空间和表。表架构保存电影数据并启用了 point-in-time 恢复。
- 使用带有 Sigv4 身份验证的安全 TLS 连接，连接到键空间。
- 查询表。添加、检索和更新电影数据。
- 更新表。添加一系列来跟踪观看的电影。
- 将表还原到以前的状态并清理资源。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
global using System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using Newtonsoft.Json;

namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
```



```
services.AddAWSService<IAmazonKeyspaces>()
    .AddTransient<KeyspacesWrapper>()
    .AddTransient<CassandraWrapper>()
)
.Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<KeyspacesBasics>();

var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load test settings from .json file.
    .AddJsonFile("settings.local.json",
        true) // Optionally load local settings.
    .Build();

var keyspacesWrapper = host.Services.GetRequiredService<KeyspacesWrapper>();
var uiMethods = new UiMethods();

var keyspaceName = configuration["KeyspaceName"];
var tableName = configuration["TableName"];

bool success; // Used to track the results of some operations.

uiMethods.DisplayOverview();
uiMethods.PressEnter();

// Create the keyspace.
var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);

// Wait for the keyspace to be available. GetKeyspace results in a
// resource not found error until it is ready for use.
try
{
    var getKeySpaceArn = "";
    Console.WriteLine($"Created {keyspaceName}. Waiting for it to become
available. ");
    do
    {
        getKeySpaceArn = await keyspacesWrapper.GetKeyspace(keyspaceName);
        Console.WriteLine(". ");
    } while (getKeySpaceArn != keyspaceArn);
}
catch (ResourceNotFoundException)
```

```
{
    Console.WriteLine("Waiting for keyspaces to be created.");
}

Console.WriteLine($"\\nThe keyspaces {keyspacesName} 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(keyspacesName, tableSchema,
tableName);

// Wait for the table to be active.
try
{
    var resp = new GetTableResponse();
    Console.WriteLine("Waiting for the new table to be active. ");
    do
    {
        try
        {
            resp = await keyspacesWrapper.GetTable(keyspacesName, tableName);
            Console.WriteLine(".");
        }
    }
}
```

```
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine(".");
        }
    } while (resp.Status != TableStatus.ACTIVE);

    // Display the table's schema.
    Console.WriteLine($"\\nTable {tableName} has been created in
{keyspaceName}");
    Console.WriteLine("Let's take a look at the schema.");
    uiMethods.DisplayTitle("All columns");
    resp.SchemaDefinition.AllColumns.ForEach(column =>
    {
        Console.WriteLine($"{column.Name, -40}\\t{column.Type, -20}");
    });

    uiMethods.DisplayTitle("Cluster keys");
    resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
    {
        Console.WriteLine($"{clusterKey.Name, -40}\\t{clusterKey.OrderBy, -20}");
    });

    uiMethods.DisplayTitle("Partition keys");
    resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
    {
        Console.WriteLine($"{partitionKey.Name}");
    });

    uiMethods.PressEnter();
    }
    catch (ResourceNotFoundException ex)
    {
        Console.WriteLine($"Error: {ex.Message}");
    }

    // Access Apache Cassandra using the Cassandra drive for C#.
    var cassandraWrapper = host.Services.GetRequiredService<CassandraWrapper>();
    var movieFilePath = configuration["MovieFile"];

    Console.WriteLine("Let's add some movies to the table we created.");
    var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);
```

```
uiMethods.PressEnter();

Console.WriteLine("Added the following movies to the table:");
var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
uiMethods.DisplayTitle("All Movies");

foreach (var row in rows)
{
    var title = row.GetValue<string>("title");
    var year = row.GetValue<int>("year");
    var plot = row.GetValue<string>("plot");
    var release_date = row.GetValue<DateTime>("release_date");
    Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
    Console.WriteLine(uiMethods.SepBar);
}

// Update the table schema
uiMethods.DisplayTitle("Update table schema");
Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

// First save the current time as a UTC Date so the original
// table can be restored later.
var timeChanged = DateTime.UtcNow;

// Now update the schema.
var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
uiMethods.PressEnter();

Console.WriteLine("Now let's mark some of the movies as watched.");

// Pick some files to mark as watched.
var movieToWatch = rows[2].GetValue<string>("title");
var watchedMovieYear = rows[2].GetValue<int>("year");
var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[6].GetValue<string>("title");
watchedMovieYear = rows[6].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);
```

```
movieToWatch = rows[9].GetValue<string>("title");
watchedMovieYear = rows[9].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[10].GetValue<string>("title");
watchedMovieYear = rows[10].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

movieToWatch = rows[13].GetValue<string>("title");
watchedMovieYear = rows[13].GetValue<int>("year");
changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

uiMethods.DisplayTitle("Watched movies");
Console.WriteLine("These movies have been marked as watched:");
rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
foreach (var row in rows)
{
    var title = row.GetValue<string>("title");
    var year = row.GetValue<int>("year");
    Console.WriteLine($"{title,-40}\t{year,8}");
}
uiMethods.PressEnter();

Console.WriteLine("We can restore the table to its previous state but that
can take up to 20 minutes to complete.");
string answer;
do
{
    Console.WriteLine("Do you want to restore the table? (y/n)");
    answer = Console.ReadLine();
} while (answer.ToLower() != "y" && answer.ToLower() != "n");

if (answer == "y")
{
    var restoredTableName = $"{tableName}_restored";
    var restoredTableArn = await keyspacesWrapper.RestoreTable(
        keyspaceName,
        tableName,
        restoredTableName,
        timeChanged);
    // Loop and call GetTable until the table is gone. Once it has been
```

```
// deleted completely, GetTable will raise a ResourceNotFoundException.
bool wasRestored = false;

try
{
    do
    {
        var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
        wasRestored = (resp.Status == TableStatus.ACTIVE);
    } while (!wasRestored);
}
catch (ResourceNotFoundException)
{
    // If the restored table raised an error, it isn't
    // ready yet.
    Console.WriteLine(".");
}
}

uiMethods.DisplayTitle("Clean up resources.");

// Delete the table.
success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
Console.WriteLine("Waiting for the table to be removed completely. ");

// Loop and call GetTable until the table is gone. Once it has been
// deleted completely, GetTable will raise a ResourceNotFoundException.
bool wasDeleted = false;

try
{
    do
    {
        var resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
    } while (!wasDeleted);
}
catch (ResourceNotFoundException ex)
{
    wasDeleted = true;
}
```

```
        Console.WriteLine($"{ex.Message} indicates that the table has been
deleted.");
    }

    // Delete the keyspace.
    success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
    Console.WriteLine("The keyspace has been deleted and the demo is now
complete.");
}
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }
}
```

```
    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }

    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
```



```
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTable(string keyspaceName, string tableName)
{
    var response = await _amazonKeyspaces.DeleteTableAsync(
        new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.GetKeyspaceAsync(
        new GetKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}

/// <summary>
/// Get information about an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the Amazon Keyspaces table.</param>
/// <returns>The response containing data about the table.</returns>
public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
{
    var response = await _amazonKeyspaces.GetTableAsync(
        new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
    return response;
}

/// <summary>
/// Lists all keyspaces for the account.
/// </summary>
/// <returns>Async task.</returns>
```

```

public async Task ListKeyspaces()
{
    var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

    Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
    Console.WriteLine(new string('-', Console.WindowWidth));
    await foreach (var keyspace in paginator.Keyspaces)
    {
        Console.WriteLine($"{keyspace.KeyspaceName, -30}\t{keyspace.ResourceArn}");
    }
}

/// <summary>
/// Lists the Amazon Keyspaces tables in a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>A list of TableSummary objects.</returns>
public async Task<List<TableSummary>> ListTables(string keyspaceName)
{
    var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
    response.Tables.ForEach(table =>
    {
        Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
    });

    return response.Tables;
}

/// <summary>
/// Restores the specified table to the specified point in time.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to restore.</param>
/// <param name="timestamp">The time to which the table will be restored.</
param>
/// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
public async Task<string> RestoreTable(string keyspaceName, string tableName,
string restoredTableName, DateTime timestamp)

```

```
{
    var request = new RestoreTableRequest
    {
        RestoreTimestamp = timestamp,
        SourceKeyspaceName = keyspaceName,
        SourceTableName = tableName,
        TargetKeyspaceName = keyspaceName,
        TargetTableName = restoredTableName
    };

    var response = await _amazonKeyspaces.RestoreTableAsync(request);
    return response.RestoredTableARN;
}

/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
        KeyspaceName = keyspaceName,
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
}
```

```
using System.Net;
using Cassandra;

namespace KeyspacesScenario;
```

```
/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
/// database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/repository/
sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        _localPathToFile = Path.GetTempPath();

        // Get the Starfield digital certificate and save it locally.
        var client = new WebClient();
        client.DownloadFile(_certLocation, $"{_localPathToFile}/{_certFileName}");

        //var httpClient = new HttpClient();
        //var httpResult = httpClient.Get(fileUrl);
        //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
        //using var fileStream = File.Create(pathToSave);
        //resultStream.CopyTo(fileStream);

        _certCollection = new X509Certificate2Collection();
    }
}
```

```

        _amazoncert = new X509Certificate2($"{_localPathToFile}/{_certFileName}");

        // Get the user name and password stored in the configuration file.
        _userName = _configuration["UserName"]!;
        _pwd = _configuration["Password"]!;

        // For a list of Service Endpoints for Amazon Keyspaces, see:
        // https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
        var awsEndpoint = _configuration["ServiceEndpoint"];

        _cluster = Cluster.Builder()
            .AddContactPoints(awsEndpoint)
            .WithPort(9142)
            .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
            .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
            .WithQueryOptions(
                new QueryOptions()
                    .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
                    .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
            .Build();
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the Apache Cassandra table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A list of movie objects.</returns>
    public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport =
0)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();

        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        // If numToImport = 0, return all movies in the collection.
        if (numToImport == 0)

```

```
    {
        // Now return the entire list of movies.
        return allMovies;
    }
    else
    {
        // Now return the first numToImport entries.
        return allMovies.GetRange(0, numToImport);
    }
}

///  

/// <summary>  

/// Insert movies into the movie table.  

/// </summary>  

/// <param name="keyspaceName">The keyspace containing the table.</param>  

/// <param name="movieTableName">The Amazon Keyspaces table.</param>  

/// <param name="movieFilePath">The path to the resource file containing  

/// movie data to insert into the table.</param>  

/// <returns>A Boolean value indicating the success of the action.</returns>  

public async Task<bool> InsertIntoMovieTable(string keyspaceName, string  

movieTableName, string movieFilePath, int numToImport = 20)  

{
    // Get some movie data from the movies.json file  

    var movies = ImportMoviesFromJson(movieFilePath, numToImport);

    var session = _cluster.Connect(keyspaceName);

    string insertCql;

    RowSet rs;

    // Now we insert the numToImport movies into the table.  

    foreach (var movie in movies)  

    {
        // Escape single quote characters in the plot.  

        insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}  

(title, year, release_date, plot) values({{movie.Title}}$, {movie.Year},  

'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}$', {{movie.Info.Plot}}$)";  

        rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
    }

    return true;
}
```

```

/// <summary>
/// Gets all of the movies in the movies table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <returns>A list of row objects containing movie data.</returns>
public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
{
    var session = _cluster.Connect();
    RowSet rs;
    try
    {
        rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

        // Extract the row data from the returned RowSet.
        var rows = rs.GetRows().ToList();
        return rows;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}

/// <summary>
/// Mark a movie in the movie table as watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <param name="title">The title of the movie to mark as watched.</param>
/// <param name="year">The year the movie was released.</param>
/// <returns>A set of rows containing the changed data.</returns>
public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
{
    var session = _cluster.Connect();
    string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
WHERE title = ${title} AND year = {year}";
    var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
    var rows = rs.GetRows().ToList();
    return rows;
}

```

```
/// <summary>
/// Retrieve the movies in the movies table where watched is true.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table.</param>
/// <returns>A list of row objects containing information about movies
/// where watched is true.</returns>
public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
tableName)
{
    var session = _cluster.Connect();
    RowSet rs;
    try
    {
        rs = await session.ExecuteAsync(new SimpleStatement($"SELECT title,
year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW FILTERING"));

        // Extract the row data from the returned RowSet.
        var rows = rs.GetRows().ToList();
        return rows;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null!;
    }
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateKeyspace](#)
 - [CreateTable](#)
 - [DeleteKeyspace](#)
 - [DeleteTable](#)
 - [GetKeyspace](#)
 - [GetTable](#)
 - [ListKeyspaces](#)

- [ListTables](#)
- [RestoreTable](#)
- [UpdateTable](#)

使用 AWS SDK for .NET 的 Kinesis 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Kinesis 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)
- [无服务器示例](#)

操作

添加标签

以下代码示例演示如何向 Kinesis 流添加标签。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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($"Tags successfully added to {streamName}.");
        }
        else
        {
            Console.WriteLine("Tags were not added to the stream.");
        }
    }

    /// <summary>
    /// Applies the set of tags to the named Kinesis stream.
    /// </summary>
    /// <param name="client">The initialized Kinesis client.</param>
    /// <param name="streamName">The name of the Kinesis stream to which
    /// the tags will be attached.</param>
    /// <param name="tags">A dictionary containing key/value pairs which
    /// will be used to create the Kinesis tags.</param>
    /// <returns>A Boolean value which represents the success or failure
    /// of AddTagsToStreamAsync.</returns>
    public static async Task<bool> ApplyTagsToStreamAsync(
        IAmazonKinesis client,
        string streamName,
        Dictionary<string, string> tags)
    {
        // Implementation of ApplyTagsToStreamAsync
    }
}
```

```
    {
        var request = new AddTagsToStreamRequest
        {
            StreamName = streamName,
            Tags = tags,
        };

        var response = await client.AddTagsToStreamAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AddTagsToStream](#) 中的。

创建流

以下代码示例演示如何创建 Kinesis 流。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateStream](#) 中的。

删除流

以下代码示例演示如何删除 Kinesis 流。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteStream](#) 中的。

取消注册用户

以下代码示例演示如何从 Kinesis 流中取消注册用户。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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)
    {
        // Implementation
    }
}
```

```
{
    var request = new DeregisterStreamConsumerRequest
    {
        StreamARN = streamARN,
        ConsumerARN = consumerARN,
        ConsumerName = consumerName,
    };

    var response = await client.DeregisterStreamConsumerAsync(request);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeregisterStreamConsumer](#) 中的。

列出流

以下代码示例演示如何列出有关一个或多个 Kinesis 流的信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListStreams](#) 中的。

列出标签

以下代码示例演示如何列出与 Kinesis 流关联的标签。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}"));
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTagsForStream](#) 中的。

列出流的使用者

以下代码示例演示如何列出 Kinesis 流的使用者。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListStreamConsumers](#) 中的。

注册用户

以下代码示例演示如何将使用者注册到 Kinesis 流。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RegisterStreamConsumer](#) 中的。

无服务器示例

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [无服务器示例](#) 存储库中查找完整示例，并了解如何进行设置和运行。

通过 .NET 将 Kinesis 事件与 Lambda 结合使用。

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
        ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
    }
}
```

```
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为从 Kinesis 流接收事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 .NET 进行 Lambda Kinesis 批处理项目失败。

```
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
```



```
if (evnt.Records.Count == 0)
{
    Logger.LogInformation("Empty Kinesis Event received");
    return new StreamsEventResponse();
}

foreach (var record in evnt.Records)
{
    try
    {
        Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
        string data = await GetRecordDataAsync(record.Kinesis, context);
        Logger.LogInformation($"Data: {data}");
        // TODO: Do interesting work based on the new data
    }
    catch (Exception ex)
    {
        Logger.LogError($"An error occurred {ex.Message}");
        /* Since we are working with streams, we can return the failed item
immediately.
        Lambda will immediately begin to retry processing from this
failed item onwards. */
        return new StreamsEventResponse
        {
            BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
            {
                new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
            }
        };
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    return new StreamsEventResponse();
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
}
```

```
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

使用 AWS SDK for .NET 的 AWS KMS 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 AWS KMS 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

为密钥创建授权

以下代码示例演示如何为 KMS 密钥创建授权。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateGrant](#) 中的。

创建密钥

以下代码示例演示如何创建 AWS KMS key。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateKey](#) 中的。

为密钥创建别名

以下代码示例演示如何为 KMS 密钥创建别名。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateAlias](#) 中的。

描述密钥

以下代码示例演示如何描述 KMS 密钥。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeKey](#) 中的。

禁用密钥

以下代码示例演示如何禁用 KMS 密钥。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DisableKey](#) 中的。

启用密钥

以下示例代码显示如何启用 KMS 密钥。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [EnableKey](#) 中的。

列出密钥的别名

以下代码示例演示如何列出 KMS 密钥的别名。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListAliases](#) 中的。

列出密钥的授权

以下代码示例演示如何列出 KMS 密钥的授权。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListGrants](#) 中的。

列出密钥

以下代码示例演示如何列出 KMS 密钥。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListKeys](#) 中的。

使用 AWS SDK for .NET 的 Lambda 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 Lambda 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Lambda

以下代码示例演示了如何开始使用 Lambda。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
        });
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFunctions](#) 中的。

主题

- [操作](#)
- [场景](#)
- [无服务器示例](#)

操作

创建函数

以下代码示例演示了如何创建 Lambda 函数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateFunction](#) 中的。

删除函数

以下代码示例演示了如何删除 Lambda 函数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteFunction](#) 中的。

获取函数

以下代码示例演示了如何获取 Lambda 函数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetFunction](#) 中的。

调用函数

以下代码示例演示了如何调用 Lambda 函数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [Invoke](#)。

列出函数

以下代码示例演示了如何列出 Lambda 函数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFunctions](#) 中的。

更新函数代码

以下代码示例演示了如何更新 Lambda 函数代码。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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}.");
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateFunctionCode](#) 中的。

更新函数配置

以下代码示例演示了如何更新 Lambda 函数配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateFunctionConfiguration](#) 中的。

场景

函数入门

以下代码示例显示了操作流程：

- 创建 IAM 角色和 Lambda 函数，然后上传处理程序代码。
- 调用单参数函数并得出结果。
- 更新函数代码并使用环境变量进行配置。
- 调用新参数函数并得出结果。显示返回的执行日志。
- 列出适用于账户的函数后再清除资源。

有关更多信息，请参阅 [使用控制台创建 Lambda 函数](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建用于执行 Lambda 操作的方法。

```
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;
    }
}
```

创建运行场景的函数。

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
```

```
using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonLambda>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<LambdaWrapper>()
                    .AddTransient<LambdaRoleWrapper>()
                    .AddTransient<UIWrapper>()
            )
            .Build();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LambdaBasics>();

        var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
        var lambdaRoleWrapper =
            host.Services.GetRequiredService<LambdaRoleWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();
    }
}
```

```
string functionName = configuration["FunctionName"]!;
string roleName = configuration["RoleName"]!;
string policyDocument = "{" +
    "\"Version\": \"2012-10-17\"," +
    "\"Statement\": [ " +
    "  {" +
    "    \"Effect\": \"Allow\"," +
    "    \"Principal\": {" +
    "      \"Service\": \"lambda.amazonaws.com\" " +
    "    }," +
    "    \"Action\": \"sts:AssumeRole\" " +
    "  }" +
    "]" +
    "}";

var incrementHandler = configuration["IncrementHandler"];
var calculatorHandler = configuration["CalculatorHandler"];
var bucketName = configuration["BucketName"];
var incrementKey = configuration["IncrementKey"];
var calculatorKey = configuration["CalculatorKey"];
var policyArn = configuration["PolicyArn"];

uiWrapper.DisplayLambdaBasicsOverview();

// Create the policy to use with the AWS Lambda functions and then attach
the
// policy to a new role.
var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

Console.WriteLine("Waiting for role to become active.");
uiWrapper.WaitABit(15, "Wait until the role is active before trying to use
it.");

// Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
var success = await lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn,
roleName);
uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to the
role.");

// Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
```

```
// (Amazon S3) bucket.
uiWrapper.DisplayTitle("Create Lambda Function");
Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
    functionName,
    bucketName,
    incrementKey,
    roleArn,
    incrementHandler);

Console.WriteLine("Waiting for the new function to be available.");
Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

// Get the Lambda function.
Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
FunctionConfiguration config;
do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.State != State.Active);

Console.WriteLine($"
The function, {functionName} has been created.");
Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

uiWrapper.PressEnter();

// List the Lambda functions.
uiWrapper.DisplayTitle("Listing all Lambda functions.");
var functions = await lambdaWrapper.ListFunctionsAsync();
DisplayFunctionList(functions);

uiWrapper.DisplayTitle("Invoke increment function");
Console.WriteLine("Now that it has been created, invoke the Lambda increment
function.");
string? value;
do
{
    Console.WriteLine("Enter a value to increment: ");
    value = Console.ReadLine();
}
while (string.IsNullOrEmpty(value));
```

```
string functionParameters = "{" +
    "\"action\": \"increment\", " +
    "\"x\": \"" + value + "\"" +
    "}";
var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
Console.WriteLine($"{value} + 1 = {answer}.");

uiWrapper.DisplayTitle("Update function");
Console.WriteLine("Now update the Lambda function code.");
await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

await lambdaWrapper.UpdateFunctionConfigurationAsync(
    functionName,
    calculatorHandler,
    new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

uiWrapper.DisplayTitle("Call updated function");
Console.WriteLine("Now call the updated function...");

bool done = false;

do
{
    string? opSelected;

    Console.WriteLine("Select the operation to perform:");
    Console.WriteLine("\t1. add");
```



```
Console.WriteLine("\t2. subtract");
Console.WriteLine("\t3. multiply");
Console.WriteLine("\t4. divide");
Console.WriteLine("\t0r enter \"q\" to quit.");
Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the operation
you want to perform: ");
do
{
    Console.Write("Your choice? ");
    opSelected = Console.ReadLine();
}
while (opSelected == string.Empty);

var operation = (opSelected) switch
{
    "1" => "add",
    "2" => "subtract",
    "3" => "multiply",
    "4" => "divide",
    "q" => "quit",
    _ => "add",
};

if (operation == "quit")
{
    done = true;
}
else
{
    // Get two numbers and an action from the user.
    value = string.Empty;
    do
    {
        Console.Write("Enter the first value: ");
        value = Console.ReadLine();
    }
    while (value == string.Empty);

    string? value2;
    do
    {
        Console.Write("Enter a second value: ");
        value2 = Console.ReadLine();
    }
}
```

```
        while (value2 == string.Empty);

        functionParameters = "{" +
            "\"action\": \"" + operation + "\", " +
            "\"x\": \"" + value + "\", " +
            "\"y\": \"" + value2 + "\"" +
            "}";

        answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"The answer when we {operation} the two numbers
is: {answer}.");
    }

    uiWrapper.PressEnter();
} while (!done);

// Delete the function created earlier.

uiWrapper.DisplayTitle("Clean up resources");
// Detach the IAM policy from the IAM role.
Console.WriteLine("First detach the IAM policy from the role.");
success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached from
the role.");

Console.WriteLine("Delete the AWS Lambda function.");
success = await lambdaWrapper.DeleteFunctionAsync(functionName);
if (success)
{
    Console.WriteLine($"The {functionName} function was deleted.");
}
else
{
    Console.WriteLine($"Could not remove the function {functionName}");
}

// Now delete the IAM role created for use with the functions
// created by the application.
Console.WriteLine("Now we can delete the role that we created.");
success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
if (success)
{
```

```
        Console.WriteLine("The role has been successfully removed.");
    }
    else
    {
        Console.WriteLine("Couldn't delete the role.");
    }

    Console.WriteLine("The Lambda Scenario is now complete.");
    uiWrapper.PressEnter();

    // Displays a formatted list of existing functions returned by the
    // LambdaMethods.ListFunctions.
    void DisplayFunctionList(List<FunctionConfiguration> functions)
    {
        functions.ForEach(functionConfig =>
        {
            Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
        });
    }
}

namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
    param>
```

```
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
    to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
    roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
    AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
    param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
    policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
    }
}
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

namespace LambdaScenarioCommon;
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM)
role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the value
passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }
}
```

```
/// <summary>
/// Display a message and wait until the user presses enter.
/// </summary>
public void PressEnter()
{
    Console.WriteLine("\nPress <Enter> to continue. ");
    _ = Console.ReadLine();
    Console.WriteLine();
}

/// <summary>
/// Pad a string with spaces to center it on the console display.
/// </summary>
/// <param name="strToCenter">The string to be centered.</param>
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);

    // Wait for the requested number of seconds.
    for (int i = numSeconds; i > 0; i--)
```

```
        {
            System.Threading.Thread.Sleep(1000);
            Console.WriteLine($"{i}...");
        }

        PressEnter();
    }
}
```

定义一个递增数字的 Lambda 处理程序。

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaIncrement;

public class Function
{
    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
    }
}
```

```
    {  
        return 0;  
    }  
}  
}
```

定义执行算术运算的第二个 Lambda 处理程序。

```
using Amazon.Lambda.Core;  
  
// Assembly attribute to enable the Lambda function's JSON input to be converted  
// into a .NET class.  
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))  
]  
  
namespace LambdaCalculator;  
  
public class Function  
{  
  
    /// <summary>  
    /// A simple function that takes two number in string format and performs  
    /// the requested arithmetic function.  
    /// </summary>  
    /// <param name="input">JSON data containing an action, and x and y values.  
    /// Valid actions include: add, subtract, multiply, and divide.</param>  
    /// <param name="context">The context object passed by Lambda containing  
    /// information about invocation, function, and execution environment.</param>  
    /// <returns>A string representing the results of the calculation.</returns>  
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext  
context)  
    {  
        var action = input["action"];  
        int x = Convert.ToInt32(input["x"]);  
        int y = Convert.ToInt32(input["y"]);  
        int result;  
        switch (action)  
        {  
            case "add":  
                result = x + y;  
                break;
```



```
        case "subtract":
            result = x - y;
            break;
        case "multiply":
            result = x * y;
            break;
        case "divide":
            if (y == 0)
            {
                Console.Error.WriteLine("Divide by zero error.");
                result = 0;
            }
            else
                result = x / y;
            break;
        default:
            Console.Error.WriteLine($"{action} is not a valid operation.");
            result = 0;
            break;
    }
    return result;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的以下主题。
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

无服务器示例

通过 Kinesis 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收因接收来自 Kinesis 流的记录而触发的事件。该函数检索 Kinesis 有效负载，将 Base64 解码，并记录下记录内容。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

通过 .NET 将 Kinesis 事件与 Lambda 结合使用。

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }
    }
}
```

```
        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收将对象上传到 S3 桶时触发的事件。该函数从事件参数中检索 S3 桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

使用 .NET 将 S3 事件与 Lambda 结合使用。

```
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);
            }
        }
    }
}
```

```
        context.Logger.LogLine($"Returning {objectResult.Key}");

        return objectResult.Key;
    }
    catch (Exception e)
    {
        context.Logger.LogLine($"Error processing request - {e.Message}");

        return string.Empty;
    }
}
}
```

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

使用 .NET 将 SNS 事件与 Lambda 结合使用。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
```

```
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收从 SQS 队列收到消息时触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

通过 .NET 将 SQS 事件与 Lambda 结合使用。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

通过 Kinesis 触发器报告 Lambda 函数批处理项目失败

以下代码示例演示如何为从 Kinesis 流接收事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 .NET 进行 Lambda Kinesis 批处理项目失败。

```
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }
    }
}
```



```
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            /* Since we are working with streams, we can return the failed item
immediately.
            Lambda will immediately begin to retry processing from this
failed item onwards. */
            return new StreamsEventResponse
            {
                BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
            {
                new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
            }
            };
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }


    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

```
public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例演示如何为从 SQS 队列接收事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

AWS SDK for .NET

 Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 .NET 进行 Lambda SQS 批处理项目失败。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
```

```
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

MediaConvert 使用示例 AWS SDK for .NET

以下代码示例向您展示了如何使用with来执行操作和实现常见场景 MediaConvert。AWS SDK for .NET

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 MediaConvert

以下代码示例演示如何开始使用 AWS Elemental MediaConvert。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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
Endpoints are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the MediaConvert endpoints.
        var response = await mediaConvertClient.DescribeEndpointsAsync(
            new DescribeEndpointsRequest()
        );

        foreach (var endPoint in response.Endpoints)
        {
            Console.WriteLine($"  \tEndPoint: {endPoint.Url}");
            Console.WriteLine();
        }
    }
}
```

```
    }  
  }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeEndpoints](#) 中的。

主题

- [操作](#)

操作

创建转码任务

以下代码示例演示如何创建 AWS Elemental MediaConvert 转码任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取端点并设置客户端。

```
// 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"];  
  
// Load the customer endpoint, if it is known.  
// When you know what your Region-specific endpoint is, set it here, or set  
it in your settings.local.json file.
```

```

var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

Console.WriteLine("Welcome to the MediaConvert Create Job example.");
// If you don't have the customer-specific endpoint, request it here.
if (string.IsNullOrEmpty(mediaConvertEndpoint))
{
    Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    AmazonMediaConvertClient client = new AmazonMediaConvertClient();
    DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
    DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
    mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
}
Console.WriteLine(new string('-', 80));
Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
Console.WriteLine(new string('-', 80));
// Because you have a service URL for MediaConvert, you don't
// need to set RegionEndpoint. If you do, the ServiceURL will
// be overwritten.
AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
{
    ServiceURL = mediaConvertEndpoint,
};

AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

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));

```

使用包装程序方法创建任务并返回任务 ID。

```

/// <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

var jobId = "";
try
{
    CreateJobResponse createJobResponse =
        await _amazonMediaConvert.CreateJobAsync(createJobRequest);

    jobId = createJobResponse.Job.Id;
}
catch (BadRequestException bre)
{
    // If the endpoint was bad.
    if (bre.Message.StartsWith("You must use the customer-"))
    {
        // The exception contains the correct endpoint; extract it.
        var mediaConvertEndpoint = bre.Message.Split('\\')[1];
        Console.WriteLine(
            $"Request failed, please use endpoint {mediaConvertEndpoint}.");
    }
    else
        throw;
}

return jobId;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateJob](#) 中的。

获取转码任务

以下代码示例演示如何获取 AWS Elemental MediaConvert 转码任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取端点并设置客户端。

```
// 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"];

// Load the customer endpoint, if it is known.
// When you know what your Region-specific endpoint is, set it here, or set
it in your settings.local.json file.
var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

Console.WriteLine("Welcome to the MediaConvert Create Job example.");
// If you don't have the customer-specific endpoint, request it here.
if (string.IsNullOrEmpty(mediaConvertEndpoint))
{
    Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    AmazonMediaConvertClient client = new AmazonMediaConvertClient();
    DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
```

```

        DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
        mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
    Console.WriteLine(new string('-', 80));
    // Because you have a service URL for MediaConvert, you don't
    // need to set RegionEndpoint. If you do, the ServiceURL will
    // be overwritten.
    AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
    {
        ServiceURL = mediaConvertEndpoint,
    };

    AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

    var wrapper = new MediaConvertWrapper(mcClient);

```

通过 ID 获取任务。

```

    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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetJob](#) 中的。

列出转码任务

以下代码示例演示如何列出 AWS Elemental MediaConvert 转码任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取端点并设置客户端。

```
    // 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"];

    // Load the customer endpoint, if it is known.
    // When you know what your Region-specific endpoint is, set it here, or set
it in your settings.local.json file.
    var mediaConvertEndpoint = _configuration["mediaConvertEndpoint"];

    Console.WriteLine("Welcome to the MediaConvert Create Job example.");
    // If you don't have the customer-specific endpoint, request it here.
    if (string.IsNullOrEmpty(mediaConvertEndpoint))
    {
        Console.WriteLine("Getting customer-specific MediaConvert endpoint.");
    }
}
```

```

        AmazonMediaConvertClient client = new AmazonMediaConvertClient();
        DescribeEndpointsRequest describeRequest = new
DescribeEndpointsRequest();
        DescribeEndpointsResponse describeResponse = await
client.DescribeEndpointsAsync(describeRequest);
        mediaConvertEndpoint = describeResponse.Endpoints[0].Url;
    }
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Using endpoint {mediaConvertEndpoint}.");
    Console.WriteLine(new string('-', 80));
    // Because you have a service URL for MediaConvert, you don't
    // need to set RegionEndpoint. If you do, the ServiceURL will
    // be overwritten.
    AmazonMediaConvertConfig mcConfig = new AmazonMediaConvertConfig
    {
        ServiceURL = mediaConvertEndpoint,
    };

    AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient(mcConfig);

    var wrapper = new MediaConvertWrapper(mcClient);

```

列出具有特定状态的任务。

```

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}.");
});

```

使用分页工具列出任务。

```

/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListJobs](#) 中的。

使用 AWS SDK for .NET 的 Organizations 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Organizations 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

将策略附加到目标

以下代码示例演示如何将 Organizations 策略附加到目标。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AttachPolicy](#) 中的。

创建策略

以下代码示例演示如何创建 Organizations 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
}
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreatePolicy](#) 中的。

创建 账户

以下代码示例演示如何创建 Organizations 账户。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

/// <summary>
/// Creates a new AWS Organizations account.
/// </summary>
public class CreateAccount
{
    /// <summary>
    /// Initializes an Organizations client object and uses it to create
    /// the new account with the name specified in accountName.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var accountName = "ExampleAccount";
        var email = "someone@example.com";

        var request = new CreateAccountRequest
        {
            AccountName = accountName,
            Email = email,
        };

        var response = await client.CreateAccountAsync(request);
        var status = response.CreateAccountStatus;

        Console.WriteLine($"The status of {status.AccountName} is
{status.State}.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateAccount](#) 中的。

创建组织

以下代码示例演示如何创建 Organizations 组织。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

/// <summary>
/// Creates an organization in AWS Organizations.
/// </summary>
public class CreateOrganization
{
    /// <summary>
    /// Creates an Organizations client object and then uses it to create
    /// a new organization with the default user as the administrator, and
    /// then displays information about the new organization.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.CreateOrganizationAsync(new
CreateOrganizationRequest
        {
            FeatureSet = "ALL",
        });

        Organization newOrg = response.Organization;
```

```
        Console.WriteLine($"Organization: {newOrg.Id} Main Account:
{newOrg.MasterAccountId}");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateOrganization](#) 中的。

创建组织单位

以下代码示例演示如何创建 Organizations 组织单位。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateOrganizationalUnit](#) 中的。

删除策略

以下代码示例演示如何删除 Organizations 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeletePolicy](#) 中的。

删除组织

以下代码示例演示如何删除 Organizations 组织。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

```
}  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteOrganization](#) 中的。

删除组织单位

以下代码示例演示如何删除 Organizations 组织单位。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using System;  
using System.Threading.Tasks;  
using Amazon.Organizations;  
using Amazon.Organizations.Model;  
  
/// <summary>  
/// Shows how to delete an existing AWS Organizations organizational unit.  
/// </summary>  
public class DeleteOrganizationalUnit  
{  
    /// <summary>  
    /// Initializes the Organizations client object and calls  
    /// DeleteOrganizationalUnitAsync to delete the organizational unit  
    /// with the selected ID.  
    /// </summary>  
    public static async Task Main()  
    {  
        // Create the client object using the default account.  
        IAmazonOrganizations client = new AmazonOrganizationsClient();  
  
        var orgUnitId = "ou-0000-000000000";  
  
        var request = new DeleteOrganizationalUnitRequest
```

```
        {
            OrganizationalUnitId = orgUnitId,
        };

        var response = await client.DeleteOrganizationalUnitAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully deleted the organizational unit
with ID: {orgUnitId}.");
        }
        else
        {
            Console.WriteLine($"Could not delete the organizational unit with
ID: {orgUnitId}.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteOrganizationalUnit](#) 中的。

从目标分离策略

以下代码示例演示如何从目标中删除 Organizations 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

/// <summary>
/// Shows how to detach a policy from an AWS Organizations organization,
```

```
/// organizational unit, or account.
/// </summary>
public class DetachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and uses it to call
    /// DetachPolicyAsync to detach the policy.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-000000000";
        var targetId = "r-0000";

        var request = new DetachPolicyRequest
        {
            PolicyId = policyId,
            TargetId = targetId,
        };

        var response = await client.DetachPolicyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully detached policy with Policy Id:
{policyId}.");
        }
        else
        {
            Console.WriteLine("Could not detach the policy.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetachPolicy](#) 中的。

列出账户

以下代码示例演示如何列出 Organizations 组织的账户。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListAccounts](#) 中的。

列出组织单位

以下代码示例演示如何列出 Organizations 组织单位。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListOrganizationalUnitsForParent](#) 中的。

列出策略

以下代码示例演示如何列出 Organizations 策略。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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 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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListPolicies](#) 中的。

使用 AWS SDK for .NET 的 Amazon Polly 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Polly 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

删除词典

以下代码示例演示如何删除 Amazon Polly 词典。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;  
    }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteLexicon](#) 中的。

获取词典

以下代码示例演示如何获取 Amazon Polly 词典。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetLexicon](#) 中的。

获取可用于合成的声音

以下代码示例演示如何获取可用于合成的 Amazon Polly 语音。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeVoices](#) 中的。

列出发音词典

以下代码示例演示如何列出 Amazon Polly 发音词典。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListLexicons](#) 中的。

存储发音词典

以下代码示例演示如何存储 Amazon Polly 发音词典。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutLexicon](#) 中的。

从文本合成语音

以下代码示例演示如何通过 Amazon Polly 从文本合成语音。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

使用 AWS SDK 在 Amazon Polly 中使用语音标记从文本中合成语音。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SynthesizeSpeech](#) 中的。

使用 AWS SDK for .NET 的 Amazon RDS 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon RDS 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon RDS

以下代码示例演示了如何开始使用 Amazon RDS。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBInstances](#)。

主题

- [操作](#)
- [场景](#)

操作

创建数据库实例

以下代码示例演示如何创建 Amazon RDS 数据库实例并等待此实例变为可用。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [CreateDBInstance](#)。

创建数据库参数组

以下代码示例演示如何创建 Amazon RDS 数据库参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ParameterGroup 中的 [CreateDB](#)。

创建数据库实例的快照

以下代码示例演示如何创建 Amazon RDS 数据库实例的快照。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [CreateDBSnapshot](#)。

删除数据库实例

以下代码示例演示如何删除 Amazon RDS 数据库实例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DeleteDBInstance](#)。

删除数据库参数组

以下代码示例演示如何删除 Amazon RDS 数据库参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ParameterGroup 中的 [deletedB](#)。

描述数据库实例

以下代码示例演示如何描述 Amazon RDS 数据库实例。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBInstances](#)。

描述数据库参数组

以下代码示例演示如何描述 Amazon RDS 数据库参数组。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ParameterGroups 中的 [describedB](#)。

描述数据库引擎版本

以下代码示例演示如何描述 Amazon RDS 数据库引擎版本。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 EngineVersions 中的 [describedB](#)。

描述数据库实例的选项

以下代码示例演示如何描述 Amazon RDS 数据库实例的选项。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 InstanceOptions 中的 [DescribeOrderable 数据库](#)。

描述数据库参数组中的参数

以下代码示例演示如何描述 Amazon RDS 数据库参数组中的参数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBParameters](#)。

描述数据库实例的快照

以下代码示例演示如何描述 Amazon RDS 数据库实例的快照。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [DescribeDBSnapshots](#)。

更新数据库参数组中的参数

以下代码示例演示如何更新 Amazon RDS 数据库参数组中的参数。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 ParameterGroup 中的 [modifyDB](#)。

场景

开始使用数据库实例

以下代码示例演示了操作流程：

- 创建自定义数据库参数组并设置参数值。
- 创建一个配置为使用参数组的数据库实例。数据库实例还包含一个数据库。
- 拍摄实例的快照。
- 删除实例和参数组。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET example performs the following tasks:
    1. Returns a list of the available DB engine families using the
    DescribeDBEngineVersionsAsync method.
    2. Selects an engine family and creates a custom DB parameter group using the
    CreateDBParameterGroupAsync method.
    3. Gets the parameter groups using the DescribeDBParameterGroupsAsync method.
    4. Gets parameters in the group using the DescribeDBParameters method.
    5. Parses and displays parameters in the group.
    6. Modifies both the auto_increment_offset and auto_increment_increment
    parameters
    using the ModifyDBParameterGroupAsync method.
    7. Gets and displays the updated parameters using the DescribeDBParameters
    method with a source of "user".
    8. Gets a list of allowed engine versions using the
    DescribeDBEngineVersionsAsync method.
```

9. Displays and selects from a list of micro instance classes available for the selected engine and version.
 10. Creates an RDS DB instance that contains a MySQL database and uses the parameter group using the `CreateDBInstanceAsync` method.
 11. Waits for DB instance to be ready using the `DescribeDBInstancesAsync` method.
 12. Prints out the connection endpoint string for the new DB instance.
 13. Creates a snapshot of the DB instance using the `CreateDBSnapshotAsync` method.
 14. Waits for DB snapshot to be ready using the `DescribeDBSnapshots` method.
 15. Deletes the DB instance using the `DeleteDBInstanceAsync` method.
 16. Waits for DB instance to be deleted using the `DescribeDbInstances` method.
 17. Deletes the parameter group using the `DeleteDBParameterGroupAsync`.
- */

```
private static readonly string sepBar = new('-', 80);
private static RDSWrapper rdsWrapper = null!;
private static ILogger logger = null!;
private static readonly string engine = "mysql";
static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon RDS service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonRDS>()
                .AddTransient<RDSWrapper>()
        )
        .Build();

    logger = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
    }).CreateLogger<RDSInstanceScenario>();

    rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

    Console.WriteLine(sepBar);
    Console.WriteLine(
```

```
        "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
        Console.WriteLine(sepBar);

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamily();

            var parameterGroup = await CreateDbParameterGroup(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParameters(parameterGroup.DBParameterGroupName, parameters);

            await DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

            var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

            var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
                instanceChoice.DBInstanceClass, newInstanceIdentifier);
            if (newInstance != null)
            {
                DisplayConnectionString(newInstance);

                await CreateSnapshot(newInstance);

                await DeleteRdsInstance(newInstance);
            }

            await DeleteParameterGroup(parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
    }
```

```
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the RDS DB parameter group family from a list of available options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamily()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

        Console.WriteLine("1. The following is a list of available DB parameter
group families:");
        int i = 1;
        var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
        foreach (var parameterGroupFamily in parameterGroupFamilies)
        {
            // List the available parameter group families.
            Console.WriteLine(
                $"{i}\t{i}. Family: {parameterGroupFamily.Key}");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
```

```
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName);

        var matchingParameters =
```



```

        parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"{p.ParameterName}." +
                $"{p.Description}." +
                $"{p.AllowedValues}." +
                $"{p.ParameterValue}"));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                int newValue = 0;
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    Int32.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }
    }
}

```

```
    }
  }

  await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

  Console.WriteLine(sepBar);
}

/// <summary>
/// Describe the user source parameters in the group.
/// </summary>
/// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
/// <returns>Async task.</returns>
public static async Task DescribeUserSourceParameters(string parameterGroupName)
{
  Console.WriteLine(sepBar);
  Console.WriteLine("7. Describe user source parameters in the group.");

  var parameters =
    await rdsWrapper.DescribeDBParameters(parameterGroupName, "user");

  parameters.ForEach(p =>
    Console.WriteLine(
      $"{p.ParameterName}." +
      $"{p.Description}." +
      $"{p.AllowedValues}." +
      $"{p.ParameterValue}."));

  Console.WriteLine(sepBar);
}

/// <summary>
/// Choose a DB engine version.
/// </summary>
/// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
/// <returns>The selected engine version.</returns>
public static async Task<DBEngineVersion> ChooseDbEngineVersion(string
dbParameterGroupFamily)
{
  Console.WriteLine(sepBar);
  // Get a list of allowed engines.
```

```

        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"{i}. Engine: {version.Engine} Version
{version.EngineVersion}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDbInstanceClass(string
engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
engineVersion);
    }

```

```
        Console.WriteLine($"8. Available micro DB instance classes for engine
{engine} and version {engineVersion}:");
        int i = 1;

        // Filter to micro instances for this example.
        allowedInstances = allowedInstances
            .Where(i => i.DBInstanceClass.Contains("micro")).ToList();

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"{i}\t{i}. Instance class: {instance.DBInstanceClass} (storage type
{instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("9. Select an available DB instance class by entering
a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB instance.</
param>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
instance.</param>
    /// <returns>The new DB instance.</returns>
```

```
public static async Task<DBInstance?> CreateRdsNewInstance(DBParameterGroup
parameterGroup,
    string engineName, string engineVersion, string instanceClass, string
instanceIdentifier)
{
    Console.WriteLine(sepBar);
    Console.WriteLine($"10. Create a new DB instance with identifier
{instanceIdentifier}.");
    bool isInstanceReady = false;
    DBInstance newInstance;
    var instances = await rdsWrapper.DescribeDBInstances();
    isInstanceReady = instances.FirstOrDefault(i =>
        i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

    if (isInstanceReady)
    {
        Console.WriteLine("Instance already created.");
        newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
    }
    else
    {
        Console.WriteLine("Please enter an admin user name:");
        var username = Console.ReadLine();

        Console.WriteLine("Please enter an admin password:");
        var password = Console.ReadLine();

        newInstance = await rdsWrapper.CreateDBInstance(
            "ExampleInstance",
            instanceIdentifier,
            parameterGroup.DBParameterGroupName,
            engineName,
            engineVersion,
            instanceClass,
            20,
            username,
            password
        );

        // 11. Wait for the DB instance to be ready.

        Console.WriteLine("11. Waiting for DB instance to be ready...");
    }
}
```

```

        while (!isInstanceReady)
        {
            instances = await
rdsWrapper.DescribeDBInstances(instanceIdentifier);
            isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
            newInstance = instances.First();
            Thread.Sleep(30000);
        }
    }

    Console.WriteLine(sepBar);
    return newInstance;
}

/// <summary>
/// Display a connection string for an RDS DB instance.
/// </summary>
/// <param name="instance">The DB instance to use to get a connection string.</
param>
public static void DisplayConnectionString(DBInstance instance)
{
    Console.WriteLine(sepBar);
    // Display the connection string.
    Console.WriteLine("12. New DB instance connection string: ");
    Console.WriteLine(
        $"{engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
"
        + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");

    Console.WriteLine(sepBar);
}

/// <summary>
/// Create a snapshot from an RDS DB instance.
/// </summary>
/// <param name="instance">DB instance to use when creating a snapshot.</param>
/// <returns>The snapshot object.</returns>
public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
{
    Console.WriteLine(sepBar);
    // Create a snapshot.
    Console.WriteLine($"13. Creating snapshot from DB instance
{instance.DBInstanceIdentifier}.");
}

```

```
        var snapshot = await
rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            var snapshots = await
rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
    {
        Console.WriteLine(sepBar);
        // Delete the DB instance.
        Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
        await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);

        // Wait for the DB instance to delete.
        Console.WriteLine($"16. Waiting for the DB instance to delete...");
        bool isInstanceDeleted = false;

        while (!isInstanceDeleted)
        {
            var instance = await rdsWrapper.DescribeDBInstances();
```

```

        isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
newInstance.DBInstanceIdentifier);
        Thread.Sleep(30000);
    }

    Console.WriteLine("DB instance deleted.");
    Console.WriteLine(sepBar);
}

/// <summary>
/// Delete a DB parameter group.
/// </summary>
/// <param name="parameterGroup">The parameter group to delete.</param>
/// <returns>Async task.</returns>
public static async Task DeleteParameterGroup(DBParameterGroup parameterGroup)
{
    Console.WriteLine(sepBar);
    // Delete the parameter group.
    Console.WriteLine($"17. Delete the DB parameter group
{parameterGroup.DBParameterGroupName}.");
    await
rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

    Console.WriteLine(sepBar);
}

```

场景用于数据库实例操作的包装程序方法。

```

/// <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;
    }
}
```

场景用于数据库参数组的包装程序方法。

```
/// <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;
}
```

场景用于数据库快照操作的包装程序方法。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateDBInstance](#)
 - [创建数据库 ParameterGroup](#)
 - [CreateDBSnapshot](#)
 - [DeleteDBInstance](#)
 - [已删除B ParameterGroup](#)
 - [describedB EngineVersions](#)
 - [DescribeDBInstances](#)
 - [describedB ParameterGroups](#)
 - [DescribeDBParameters](#)
 - [DescribeDBSnapshots](#)
 - [DescribeOrderable数据库 InstanceOptions](#)
 - [modifyDB ParameterGroup](#)

使用 AWS SDK for .NET 的 Amazon Rekognition 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Rekognition 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

将图像中的人脸与参照图像进行对比

以下代码示例演示如何使用 Amazon Rekognition 将图像中的人脸与参照图像进行对比。

有关更多信息，请参阅[比较图像中的人脸](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CompareFaces](#) 中的。

创建集合

以下代码示例演示如何创建 Amazon Rekognition 集合。

有关更多信息，请参阅 [创建集合](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateCollection](#) 中的。

删除集合

以下代码示例演示如何删除 Amazon Rekognition 集合。

有关更多信息，请参阅 [删除集合](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteCollection](#) 中的。

从集合中删除人脸

以下代码示例演示如何从 Amazon Rekognition 集合中删除人脸。

有关更多信息，请参阅 [从集合中删除人脸](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
        });
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteFaces](#) 中的。

描述集合

以下代码示例演示如何描述 Amazon Rekognition 集合。

有关更多信息，请参阅 [描述集合](#)。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
```

```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeCollection](#) 中的。

检测图像中的人脸

以下代码示例演示如何使用 Amazon Rekognition 检测图像中的人脸。

有关更多信息，请参阅 [检测图像中的人脸](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

```
}
```

显示图像中所有人脸的边界框信息。

```
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}");  
    }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectFaces](#) 中的。

检测图像中的标签

以下代码示例演示如何使用 Amazon Rekognition 检测图像中的标签。

有关更多信息，请参阅 [检测图像中的标签](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

检测存储在计算机上的图像文件中的标签。

```
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);
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectLabels](#) 中的。

检测图像中的审核标签

以下代码示例演示如何通过 Amazon Rekognition 检测图像中的审核标签。审核标签可识别可能不适合某些受众的内容。

有关更多信息，请参阅 [检测不适宜的图像](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectModerationLabels](#) 中的。

检测图像中的文本

以下代码示例演示如何使用 Amazon Rekognition 检测图像中的文本。

有关更多信息，请参阅 [检测图像中的文本](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DetectText](#) 中的。

获取有关名人的信息

以下代码示例演示如何使用 Amazon Rekognition 获取有关名人的信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
        });
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetCelebrityInfo](#) 中的。

将人脸索引到集合中

以下代码示例演示如何对图像中的人脸编制索引并将其添加到 Amazon Rekognition 集合中。

有关更多信息，请参阅[将人脸添加到集合中](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [IndexFaces](#) 中的。

列出集合

以下代码示例演示如何列出 Amazon Rekognition 集合。

有关更多信息，请参阅 [列出集合](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListCollections](#) 中的。

列出集合中的人脸

以下代码示例演示如何列出 Amazon Rekognition 集合中的人脸。

有关更多信息，请参阅 [列出集合中的人脸](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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));
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFaces](#) 中的。

识别图像中的名人

以下代码示例演示如何通过 Amazon Rekognition 识别图像中的名人。

有关更多信息，请参阅 [识别图像中的名人](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Shows how to use Amazon Rekognition to identify celebrities in a photo.
/// </summary>
```

```
public class CelebritiesInImage
{
    public static async Task Main(string[] args)
    {
        string photo = "moviestars.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

        var img = new Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
            data = new byte[fs.Length];
            fs.Read(data, 0, (int)fs.Length);
        }
        catch (Exception)
        {
            Console.WriteLine($"Failed to load file {photo}");
            return;
        }

        img.Bytes = new MemoryStream(data);
        recognizeCelebritiesRequest.Image = img;

        Console.WriteLine($"Looking for celebrities in image {photo}\n");

        var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

        Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
        recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
        {
            Console.WriteLine($"Celebrity recognized: {celeb.Name}");
            Console.WriteLine($"Celebrity ID: {celeb.Id}");
            BoundingBox boundingBox = celeb.Face.BoundingBox;
            Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
            Console.WriteLine("Further information (if available):");
            celeb.UrlsWithMetadata.ForEach(url =>
```

```
        {
            Console.WriteLine(url);
        });
    });

    Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RecognizeCelebrities](#) 中的。

搜索集合中的人脸

以下代码示例演示如何在 Amazon Rekognition 集合中搜索与该集合中另一张人脸匹配的人脸。

有关更多信息，请参阅 [搜索人脸 \(人脸 ID\)](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    });
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SearchFaces](#) 中的。

对比参照图像在集合中搜索人脸

以下代码示例演示如何对比参照图像在 Amazon Rekognition 集合中搜索人脸。

有关更多信息，请参阅 [搜索人脸 \(图像\)](#)。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    });
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SearchFacesByImage](#) 中的。

使用 AWS SDK for .NET 的 Route 53 域注册示例

以下代码示例演示如何将 AWS SDK for .NET 与 Route 53 域注册结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Route 53 域注册

以下代码示例显示如何开始使用 Route 53 域注册。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
public static class HelloRoute53Domains
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        // the Amazon Route 53 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRoute53Domains>()
            ).Build();

        // Now the client is available for injection.
        var route53Client =
            host.Services.GetRequiredService<IAmazonRoute53Domains>();

        // You can use await and any of the async methods to get a response.
        var response = await route53Client.ListPricesAsync(new ListPricesRequest
            { Tld = "com" });
        Console.WriteLine($"Hello Amazon Route 53 Domains! Following are prices
            for .com domain operations:");
        var comPrices = response.Prices.FirstOrDefault();
        if (comPrices != null)
        {
            Console.WriteLine($"Registration: {comPrices.RegistrationPrice?.Price}
                {comPrices.RegistrationPrice?.Currency}");
            Console.WriteLine($"Renewal: {comPrices.RenewalPrice?.Price}
                {comPrices.RenewalPrice?.Currency}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListPrices](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

检查域可用性

以下代码示例演示如何检查域的可用性。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CheckDomainAvailability](#) 中的。

检查域可转移性

以下代码示例演示如何检查域的可转移性。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CheckDomainTransferability](#) 中的。

获取域详细信息

以下代码示例演示如何获取域的详细信息。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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.";
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetDomainDetail](#) 中的。

获取操作详细信息

以下代码示例演示如何获取操作的详细信息。

AWS SDK for .NET

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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\n" +
            $"{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}.";
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetOperationDetail](#) 中的。

获取建议的域名

以下代码示例演示如何获取域名建议。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetDomainSuggestions](#) 中的。

列出域价格

以下代码示例演示如何列出域价格。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListPrices](#) 中的。

列出域

以下代码示例演示如何列出已注册的域。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListDomains](#) 中的。

列表操作

以下代码示例演示如何列出操作。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListOperations](#) 中的。

注册域

以下代码示例演示如何注册域。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RegisterDomain](#) 中的。

查看账单

以下代码示例演示如何查看账单记录。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ViewBilling](#) 中的。

场景

域入门

以下代码示例演示了操作流程：

- 列出当前域，并列过去一年的操作。
- 查看过去一年的账单，并查看域类型的价格。
- 获取域建议。
- 检查域可用性和可转移性。
- (可选) 申请域注册。
- 获取操作详细信息。
- (可选) 获取域详细信息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
public static class Route53DomainScenario
{
    /*
    Before running this .NET code example, set up your development environment,
    including your credentials.

    This .NET example performs the following tasks:
    1. List current domains.
    2. List operations in the past year.
    3. View billing for the account in the past year.
    4. View prices for domain types.
    5. Get domain suggestions.
    6. Check domain availability.
    7. Check domain transferability.
```

```
        8. Optionally, request a domain registration.
        9. Get an operation detail.
        10. Optionally, get a domain detail.
    */

    private static Route53Wrapper _route53Wrapper = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRoute53Domains>()
                    .AddTransient<Route53Wrapper>()
            )
            .Build();

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

        var logger = LoggerFactory.Create(builder =>
            {
                builder.AddConsole();
            }).CreateLogger(typeof(Route53DomainScenario));

        _route53Wrapper = host.Services.GetRequiredService<Route53Wrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon Route 53 domains example
scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
```

```
        await ListDomains();
        await ListOperations();
        await ListBillingRecords();
        await ListPrices();
        await ListDomainSuggestions();
        await CheckDomainAvailability();
        await CheckDomainTransferability();
        var operationId = await RequestDomainRegistration();
        await GetOperationalDetail(operationId);
        await GetDomainDetails();
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "There was a problem executing the scenario.");
    }

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("The Amazon Route 53 domains example scenario is
complete.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List account registered domains.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDomains()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"1. List account domains.");
    var domains = await _route53Wrapper.ListDomains();
    for (int i = 0; i < domains.Count; i++)
    {
        Console.WriteLine($"{i + 1}. {domains[i].DomainName}");
    }

    if (!domains.Any())
    {
        Console.WriteLine("\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($"6. Check domain availability.");
    string? domainName = null;
    while (domainName == null || string.IsNullOrEmpty(domainName))
    {
        Console.WriteLine($"Enter a domain name to check domain availability.");
        domainName = Console.ReadLine();
    }

    var availability = await
_route53Wrapper.CheckDomainAvailability(domainName);
    Console.WriteLine($"\\tAvailability: {availability}");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Check transferability for a domain name.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CheckDomainTransferability()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"7. Check domain transferability.");
    string? domainName = null;
    while (domainName == null || string.IsNullOrEmpty(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(
            $"{\t}Registration requested. Operation Id: {operationId}");
    }

    return operationId;
}

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

/// <summary>
/// Get details for an operation.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetOperationalDetail(string? operationId)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Get an operation detail.");

    var operationDetails =
        await _route53Wrapper.GetOperationDetail(operationId);

    Console.WriteLine(operationDetails);

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

/// <summary>
/// Optionally, get details for a registered domain.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string?> GetDomainDetails()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"10. Get details on a domain.");

    Console.WriteLine($"{\t}Note: 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;
    }
}
```

场景用于 Route 53 域注册操作的包装程序方法。

```
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 = $"{\t}Operation {operationId}:\n" +
            $"{\t}For domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}. \n" +
            $"{\t}Message is {operationDetails.Message}. \n" +
            $"{\t}Status 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.";
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CheckDomainAvailability](#)
 - [CheckDomainTransferability](#)
 - [GetDomainDetail](#)
 - [GetDomainSuggestions](#)
 - [GetOperationDetail](#)
 - [ListDomains](#)
 - [ListOperations](#)
 - [ListPrices](#)
 - [RegisterDomain](#)
 - [ViewBilling](#)

使用 AWS SDK for .NET 的 Amazon S3 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 Amazon S3 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)
- [场景](#)
- [无服务器示例](#)

操作

将 CORS 规则添加到桶

以下代码示例演示了如何向 S3 存储桶添加跨源资源共享 (CORS) 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketCors](#) 中的。

向桶添加生命周期配置

以下代码示例演示如何将生命周期配置添加到 S3 桶。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketLifecycleConfiguration](#) 中的。

取消分段上传

以下代码示例说明如何取消分段上传。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AbortMultipartUploads](#) 中的。

复制对象到不同存储桶

以下代码示例演示了如何将 S3 对象从一个桶复制到另一个桶。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CopyObject](#) 中的。

创建存储桶

以下代码示例演示了如何创建 S3 存储桶。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateBucket](#) 中的。

从桶中删除 CORS 规则

以下代码示例演示如何从 S3 桶中删除 CORS 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteBucketCors](#) 中的。

删除空存储桶

以下代码示例演示了如何删除空的 S3 存储桶。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteBucket](#) 中的。

删除对象

以下代码示例演示了如何删除 S3 对象。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

删除不受版本控制的 S3 桶中的对象。

```
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.");
        }
    }
}
```

删除受版本控制的 S3 桶中的对象。

```
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 to create.</
param>
    /// <returns>The Version ID of the created object.</returns>
    public static async Task<string> PutAnObject(IAmazonS3 client, string
bucketName, string objectKey)
    {
        PutObjectRequest request = new PutObjectRequest()
        {
            BucketName = bucketName,
            Key = objectKey,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = await client.PutObjectAsync(request);
        return response.VersionId;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteObject](#) 中的。

删除多个对象

以下代码示例演示了如何从 S3 存储桶中删除多个对象。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

从 S3 桶中删除所有对象。

```
/// <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;
    }
}
```

删除不受版本控制的 S3 桶中的多个对象。

```
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;
}
}
```

删除受版本控制的 S3 桶中的多个对象。

```
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 works in
an
/// Amazon S3 bucket with versioning enabled.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// PutObjectAsync to create temporary objects for the example.</param>
/// <param name="bucketName">A string representing the name of the S3
/// bucket where we will create the temporary objects.</param>
/// <param name="number">The number of temporary objects to create.</param>
/// <returns>A list of the KeyVersion objects.</returns>
private static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3
client, string bucketName, int number)
{
    var keys = new List<KeyVersion>();

    for (var i = 0; i < number; i++)
    {
        string key = "ObjectToDelete-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        var response = await client.PutObjectAsync(request);
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
            VersionId = response.VersionId,
        };

        keys.Add(keyVersion);
    }
}
```



```
        return keys;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteObjects](#) 中的。

删除桶的生命周期配置

以下代码示例显示如何删除 S3 桶的生命周期配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteBucketLifecycle](#) 中的。

启用日志记录

以下代码示例演示了如何使用在 S3 桶上启用日志记录。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketLogging](#) 中的。

启用通知

下面的代码示例显示如何对 S3 桶启用通知。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketNotificationConfiguration](#) 中的。

启用传输加速

以下代码示例显示如何对 S3 桶启用传输加速。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketAccelerateConfiguration](#) 中的。

获取存储桶的 CORS 规则

以下代码示例演示了如何获取 S3 存储桶的跨源资源共享 (CORS) 规则。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetBucketCors](#) 中的。

从存储桶中获取对象

以下代码示例演示了如何从 S3 存储桶中的对象读取数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetObject](#) 中的。

获取存储桶的 ACL

以下代码示例演示了如何获取 S3 存储桶的访问控制列表 (ACL)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetBucketAcl](#) 中的。

获取桶的生命周期配置

以下代码示例显示如何获取 S3 桶的生命周期配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
    /// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetBucketLifecycleConfiguration](#) 中的。

获取存储桶网站的配置

以下代码示例演示了如何获取 S3 存储桶的网站配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
// 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}");
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetBucketWebsite](#) 中的。

列出存储桶

以下代码示例演示了如何列出 S3 存储桶。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListBuckets](#) 中的。

列出桶中的对象版本

以下代码示例显示如何列出 S3 桶中的对象版本。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
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}'");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListObjectVersions](#) 中的。

列出存储桶中的对象

以下代码示例演示了如何列出 S3 存储桶中的对象。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
}
```

使用分页工具列出对象。

```
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}");
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考中的 [ListObjectsV2](#)。

还原对象的存档副本

以下代码示例演示如何将对象的存档副本还原到 S3 桶。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [RestoreObject](#) 中的。

为存储桶设置新 ACL

以下代码示例演示了如何为 S3 存储桶设置新的访问控制列表 (ACL)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

    /// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketAcl](#) 中的。

设置存储桶的网站配置

以下示例演示了如何设置 S3 存储桶的网站配置。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
// Put the website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
{
    BucketName = bucketName,
```

```
        WebsiteConfiguration = new WebsiteConfiguration()
        {
            IndexDocumentSuffix = indexDocumentSuffix,
            ErrorDocument = errorDocument,
        },
    };
    PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutBucketWebsite](#) 中的。

将对象上传到存储桶

以下代码示例演示了如何将对象上传到 S3 存储桶。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
        }
    }
}
```

使用服务器端加密上载对象。

```
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");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [PutObject](#) 中的。

场景

创建预签名 URL

以下代码示例演示了如何为 Amazon S3 创建预签名 URL 以及如何上传对象。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

生成可在有限时间内执行 Amazon S3 操作的预签名 URL。

```
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
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;
    }
}
```

生成预签名 URL 并使用该 URL 执行上载。

```
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 HttpResponseMessage.</returns>
public static async Task<bool> UploadObject(string filePath, string url)
{
    using var streamContent = new StreamContent(
        new FileStream(filePath, FileMode.Open, FileAccess.Read));

    var response = await httpClient.PutAsync(url, streamContent);
    return response.IsSuccessStatusCode;
}

/// <summary>
/// Generates a presigned URL which will be used to upload an object to
/// an Amazon S3 bucket.
/// </summary>
/// <param name="client">The initialized Amazon S3 client object used to
call
/// GetPreSignedURL.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket to which the
param>
/// 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;
}
```

```
}  
}
```

开始使用存储桶和对象

以下代码示例显示了操作流程：

- 创建存储桶并上传文件。
- 下载存储桶中的对象。
- 复制对象到存储桶中的子文件夹。
- 列出存储桶中的对象。
- 删除存储桶及其对象。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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);
```

```
}  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

加密入门

下面的代码示例显示如何开始 Amazon S3 对象的加密。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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>    /// <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);  
    }  
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CopyObject](#)
 - [GetObject](#)
 - [GetObjectMetadata](#)

标签入门

以下代码示例演示如何开始为 Amazon S3 对象加标签。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}'");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetObjectTagging](#) 中的。

管理访问控制列表 (ACL)

以下代码示例显示如何管理 Amazon S3 桶的访问控制列表 (ACL)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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,  
    });  
}  
  
}
```

• 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。

- [GetBucketAcl](#)
- [GetObjectAcl](#)
- [PutBucketAcl](#)
- [PutObjectAcl](#)

执行分段复制

以下代码示例显示如何执行 Amazon S3 对象的分段复制。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
    }
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CompleteMultipartUpload](#)
 - [CreateMultipartUpload](#)
 - [GetObjectMetadata](#)
 - [UploadPartCopy](#)

上传或下载大文件

以下代码示例演示了如何向 Amazon S3 上传大文件或从 Amazon S3 下载大文件。

有关更多信息，请参阅[使用分段上传操作上传对象](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用使用 Amazon S3 将文件传输到 S3 存储桶和传出 S3 存储桶的函数 `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);
}
```

上载单个文件。

```
/// <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;
    }
}

```

上载整个本地目录。

```

/// <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;
        }
    }
}
```

下载单个文件。

```
/// <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}"));
}
```

下载 S3 桶的内容。

```
/// <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;
    }
```

使用跟踪上传进度 `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}");
    }
}
```

上载经过加密的对象。

```
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);
    }
}
}
```

无服务器示例

通过 Amazon S3 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收将对象上传到 S3 桶时触发的事件。该函数从事件参数中检索 S3 桶名称和对象密钥，并调用 Amazon S3 API 来检索和记录对象的内容类型。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

使用 .NET 将 S3 事件与 Lambda 结合使用。

```
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                }
            }
        }
    }
}
```

```
        return string.Empty;
    }

    var bucket = evt.Records[0].S3.Bucket.Name;
    var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

    context.Logger.LogLine($"Request is for {bucket} and {key}");

    var objectResult = await _s3Client.GetObjectAsync(bucket, key);

    context.Logger.LogLine($"Returning {objectResult.Key}");

    return objectResult.Key;
}
catch (Exception e)
{
    context.Logger.LogLine($"Error processing request - {e.Message}");

    return string.Empty;
}
}
}
```

使用 AWS SDK for .NET 的 S3 Glacier 示例

以下代码示例演示如何将 AWS SDK for .NET 与 S3 Glacier 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。


每个示例都包含一个指向的链接 [GitHub](#)，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon S3 Glacier

以下代码示例演示了如何开始使用 Amazon S3 Glacier。

AWS SDK for .NET

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListVaults](#) 中的。

主题

- [操作](#)

操作

添加标签

以下代码示例演示了如何添加标签到 Amazon S3 Glacier 保管库中。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AddTagsToVault](#) 中的。

创建文件库

以下代码示例演示了如何创建 Amazon S3 Glacier 文件库。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateVault](#) 中的。

描述保管库

以下代码示例演示了如何描述 Amazon S3 Glacier 保管库。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeVault](#) 中的。

下载存档

以下代码示例演示如何下载 Amazon S3 Glacier 存档。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

此示例使用 `ArchiveTransferManager` 类。有关 API 的详细信息，请参阅 [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}%");
        }
    }
}
```

列出任务

以下代码示例演示如何列出 Amazon S3 Glacier 任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListJobs](#) 中的。

列出标签

以下代码示例演示了如何列出 Amazon S3 Glacier 保管库的标签。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTagsForVault](#) 中的。

列出保管库

以下代码示例演示如何列出 Amazon S3 Glacier 文件库。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListVaults](#) 中的。

将存档上传到保管库

以下代码示例演示了如何将存档上传到 Amazon S3 Glacier 文件库。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UploadArchive](#) 中的。

SageMaker 使用示例 AWS SDK for .NET

以下代码示例向您展示了如何使用 `with` 来执行操作和实现常见场景 SageMaker。AWS SDK for .NET

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

你好 SageMaker

以下代码示例演示了如何开始使用 SageMaker。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using Amazon.SageMaker;
using Amazon.SageMaker.Model;

namespace SageMakerActions;

public static class HelloSageMaker
{
    static async Task Main(string[] args)
    {
        var sageMakerClient = new AmazonSageMakerClient();

        Console.WriteLine($"Hello Amazon SageMaker! Let's list some of your notebook
instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five notebook instances.
        var response = await sageMakerClient.ListNotebookInstancesAsync(
            new ListNotebookInstancesRequest()
            {
                MaxResults = 5
            });

        if (!response.NotebookInstances.Any())
        {
            Console.WriteLine($"No notebook instances found.");
            Console.WriteLine("See https://docs.aws.amazon.com/sagemaker/latest/dg/
howitworks-create-ws.html to create one.");
        }

        foreach (var notebookInstance in response.NotebookInstances)
        {
            Console.WriteLine($"Instance:
{notebookInstance.NotebookInstanceName}");
            Console.WriteLine($"Arn: {notebookInstance.NotebookInstanceArn}");
            Console.WriteLine($"Creation Date:
{notebookInstance.CreationTime.ToShortDateString()}");
            Console.WriteLine();
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListNotebookInstances](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建管道

以下代码示例显示了如何在 SageMaker 中创建或更新管道。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreatePipeline](#)
 - [UpdatePipeline](#)

删除管道

以下代码示例说明如何删除中的管道 SageMaker。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeletePipeline](#) 中的。

描述管道执行

以下代码示例说明如何描述中的管道执行 SageMaker。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribePipelineExecution](#) 中的。

执行管道

以下代码示例显示了如何在中启动管道执行 SageMaker。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartPipelineExecution](#) 中的。

场景

开始使用地理空间任务和管道

以下代码示例演示了操作流程：

- 为管道设置资源。
- 设置用于执行地理空间任务的管道。
- 启动管道执行。
- 监控执行状态。
- 查看管道输出。
- 清理资源。

有关更多信息，请参阅[在 Community.aws 上使用软件开发工具包创建和运行 SageMaker 管道](#)。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建一个封装 SageMaker 操作的类。

```
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;
    }
}

```

创建一个处理来自 SageMaker 管道的回调的函数。

```

using System.Text.Json;
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

// Assembly attribute to enable the AWS Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SageMakerLambda;

/// <summary>
/// The AWS Lambda function handler for the Amazon SageMaker pipeline.
/// </summary>
public class SageMakerLambdaFunction
{

```

```
    /// <summary>
    /// Default constructor. This constructor is used by AWS Lambda to construct the
    instance. When invoked in a Lambda environment
    /// the AWS credentials will come from the AWS Identity and Access Management
    (IAM) role associated with the function. The AWS Region will be set to the
    /// Region that the Lambda function is running in.
    /// </summary>
    public SageMakerLambdaFunction()
    {
    }

    /// <summary>
    /// The AWS Lambda function handler that processes events from the SageMaker
    pipeline and starts a job or export.
    /// </summary>
    /// <param name="request">The custom SageMaker pipeline request object.</param>
    /// <param name="context">The Lambda context.</param>
    /// <returns>The dictionary of output parameters.</returns>
    public async Task<Dictionary<string, string>> FunctionHandler(PipelineRequest
    request, ILambdaContext context)
    {
        var geoSpatialClient = new AmazonSageMakerGeospatialClient();
        var sageMakerClient = new AmazonSageMakerClient();
        var responseDictionary = new Dictionary<string, string>();
        context.Logger.LogInformation("Function handler started with request: " +
    JsonSerializer.Serialize(request));
        if (request.Records != null && request.Records.Any())
        {
            context.Logger.LogInformation("Records found, this is a queue event.
    Processing the queue records.");
            foreach (var message in request.Records)
            {
                await ProcessMessageAsync(message, context, geoSpatialClient,
    sageMakerClient);
            }
        }
        else if (!string.IsNullOrEmpty(request.vej_export_config))
        {
            context.Logger.LogInformation("Export configuration found, this is an
    export. Start the Vector Enrichment Job (VEJ) export.");

            var outputConfig =
                JsonSerializer.Deserialize<ExportVectorEnrichmentJobOutputConfig>(
                    request.vej_export_config);
        }
    }
}
```

```
        var exportResponse = await
geoSpatialClient.ExportVectorEnrichmentJobAsync(
    new ExportVectorEnrichmentJobRequest()
    {
        Arn = request.vej_arn,
        ExecutionRoleArn = request.Role,
        OutputConfig = outputConfig
    });
    context.Logger.LogInformation($"Export response:
{JsonSerializer.Serialize(exportResponse)}");
    responseDictionary = new Dictionary<string, string>
    {
        { "export_eoj_status", exportResponse.ExportStatus.ToString() },
        { "vej_arn", exportResponse.Arn }
    };
}
else if (!string.IsNullOrEmpty(request.vej_name))
{
    context.Logger.LogInformation("Vector Enrichment Job name found,
starting the job.");
    var inputConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobInputConfig>(
            request.vej_input_config);

    var jobConfig =
        JsonSerializer.Deserialize<VectorEnrichmentJobConfig>(
            request.vej_config);

    var jobResponse = await geoSpatialClient.StartVectorEnrichmentJobAsync(
        new StartVectorEnrichmentJobRequest()
        {
            ExecutionRoleArn = request.Role,
            InputConfig = inputConfig,
            Name = request.vej_name,
            JobConfig = jobConfig
        });
    context.Logger.LogInformation("Job response: " +
JsonSerializer.Serialize(jobResponse));
    responseDictionary = new Dictionary<string, string>
    {
        { "vej_arn", jobResponse.Arn },
        { "statusCode", jobResponse.HttpStatusCode.ToString() }
```

```
        };
    }
    return responseDictionary;
}

/// <summary>
/// Process a queue message and check the status of a SageMaker job.
/// </summary>
/// <param name="message">The queue message.</param>
/// <param name="context">The Lambda context.</param>
/// <param name="geoClient">The SageMaker GeoSpatial client.</param>
/// <param name="sageMakerClient">The SageMaker client.</param>
/// <returns>Async task.</returns>
private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context,
    AmazonSageMakerGeospatialClient geoClient, AmazonSageMakerClient
sageMakerClient)
{
    context.Logger.LogInformation($"Processed message {message.Body}");

    // Get information about the SageMaker job.
    var payload = JsonSerializer.Deserialize<QueuePayload>(message.Body);
    context.Logger.LogInformation($"Payload token {payload!.token}");
    var token = payload.token;

    if (payload.arguments.ContainsKey("vej_arn"))
    {
        // Use the job ARN and the token to get the job status.
        var job_arn = payload.arguments["vej_arn"];
        context.Logger.LogInformation($"Token: {token}, arn {job_arn}");

        var jobInfo = geoClient.GetVectorEnrichmentJobAsync(
            new GetVectorEnrichmentJobRequest()
            {
                Arn = job_arn
            });
        context.Logger.LogInformation("Job info: " +
            JsonSerializer.Serialize(jobInfo));
        if (jobInfo.Result.Status == VectorEnrichmentJobStatus.COMPLETED)
        {
            context.Logger.LogInformation($"Status completed, resuming
            pipeline...");
            await sageMakerClient.SendPipelineExecutionStepSuccessAsync(
                new SendPipelineExecutionStepSuccessRequest()
            );
        }
    }
}
```



```
        {
            CallbackToken = token,
            OutputParameters = new List<OutputParameter>()
            {
                new OutputParameter()
                    { Name = "export_status", Value =
jobInfo.Result.Status }
            }
        });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.FAILED)
    {
        context.Logger.LogInformation($"Status failed, stopping
pipeline...");
        await sageMakerClient.SendPipelineExecutionStepFailureAsync(
            new SendPipelineExecutionStepFailureRequest()
            {
                CallbackToken = token,
                FailureReason = jobInfo.Result.ErrorDetails.ErrorMessage
            });
    }
    else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.IN_PROGRESS)
    {
        // Put this message back in the queue to reprocess later.
        context.Logger.LogInformation(
            $"Status still in progress, check back later.");
        throw new("Job still running.");
    }
}
}
```

在命令提示符中运行交互式场景。

```
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($"Queue ready with Url {response.QueueUrl}.");
    Console.WriteLine(new string('-', 80));
    return response.QueueUrl;
}

}

/// <summary>
/// Connect the queue to the Lambda function as an event source.
/// </summary>
/// <param name="queueUrl">The URL for the queue.</param>
/// <returns>Async task.</returns>
public static async Task ConnectLambda(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Connecting the Lambda function and queue for the
pipeline.");
}
```

```
var queueAttributes = await _sqsClient.GetQueueAttributesAsync(
    new GetQueueAttributesRequest() { QueueUrl = queueUrl, AttributeNames =
new List<string>() { "All" } });
var queueArn = queueAttributes.QueueARN;

var eventSource = await _lambdaClient.ListEventSourceMappingsAsync(
    new ListEventSourceMappingsRequest()
    {
        FunctionName = lambdaFunctionName
    });

if (!eventSource.EventSourceMappings.Any())
{
    // Only add the event source mapping if it does not already exist.
    await _lambdaClient.CreateEventSourceMappingAsync(
        new CreateEventSourceMappingRequest()
        {
            EventSourceArn = queueArn,
            FunctionName = lambdaFunctionName,
            Enabled = true
        });
}

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

/// <summary>
/// Set up the bucket to use for pipeline input and output.
/// </summary>
/// <param name="bucketName">The name for the bucket.</param>
/// <returns>Async task.</returns>
public static async Task SetupBucket(string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up bucket {bucketName}.");

    var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
        bucketName);

    if (!bucketExists)
    {
        await _s3Client.PutBucketAsync(new PutBucketRequest()
        {
```

```
        BucketName = bucketName,
        BucketRegion = S3Region.USWest2
    });

    Thread.Sleep(5000);

    await _s3Client.PutObjectAsync(new PutObjectRequest()
    {
        BucketName = bucketName,
        Key = "samplefiles/latlongtest.csv",
        FilePath = "latlongtest.csv"
    });
}

Console.WriteLine($"\\tBucket {bucketName} ready.");
Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Display some results from the output directory.
/// </summary>
/// <param name="bucketName">The name for the bucket.</param>
/// <returns>Async task.</returns>
public static async Task<string> GetOutputResults(string bucketName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Getting output results {bucketName}.");
    string outputKey = "";
    Thread.Sleep(15000);
    var outputFiles = await _s3Client.ListObjectsAsync(
        new ListObjectsRequest()
        {
            BucketName = bucketName,
            Prefix = "outputfiles/"
        });

    if (outputFiles.S3Objects.Any())
    {
        var sampleOutput = outputFiles.S3Objects.OrderBy(s =>
s.LastModified).Last();
        Console.WriteLine($"\\tOutput file: {sampleOutput.Key}");
        var outputSampleResponse = await _s3Client.GetObjectAsync(
            new GetObjectRequest()
            {
```

```

        BucketName = bucketName,
        Key = sampleOutput.Key
    });
    outputKey = sampleOutput.Key;
    StreamReader reader = new
StreamReader(outputSampleResponse.ResponseStream);
    await reader.ReadLineAsync();
    Console.WriteLine("\tOutput file contents: \n");
    for (int i = 0; i < 10; i++)
    {
        if (!reader.EndOfStream)
        {
            Console.WriteLine("\t" + await reader.ReadLineAsync());
        }
    }
}

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

/// <summary>
/// Create a pipeline from the example pipeline JSON
/// that includes the Lambda, callback, processing, and export jobs.
/// </summary>
/// <param name="roleArn">The ARN of the role for the pipeline.</param>
/// <param name="functionArn">The ARN of the Lambda function for the pipeline.</
param>
/// <param name="pipelineName">The name for the pipeline.</param>
/// <returns>The ARN of the pipeline.</returns>
public static async Task<string> SetupPipeline(string roleArn, string
functionArn, string pipelineName)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Setting up the pipeline.");

    var pipelineJson = await File.ReadAllTextAsync("GeoSpatialPipeline.json");

    // Add the correct function ARN instead of the placeholder.
    pipelineJson = pipelineJson.Replace("*FUNCTION_ARN*", functionArn);

    var pipelineArn = await _sageMakerWrapper.SetupPipeline(pipelineJson,
roleArn, pipelineName,
        "sdk example pipeline", pipelineName);
}

```

```
        Console.WriteLine($"\\tPipeline set up with ARN {pipelineArn}.");
        Console.WriteLine(new string('-', 80));

        return pipelineArn;
    }

    /// <summary>
    /// Start a pipeline run with job configurations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue used in the pipeline.</param>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>The pipeline run ARN.</returns>
    public static async Task<string> ExecutePipeline(
        string queueUrl,
        string roleArn,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Starting pipeline execution.");

        var input = $"s3://{bucketName}/samplefiles/latlongtest.csv";
        var output = $"s3://{bucketName}/outputfiles/";

        var executionARN =
            await _sageMakerWrapper.ExecutePipeline(queueUrl, input, output,
                pipelineName, roleArn);

        Console.WriteLine($"\\tRun started with ARN {executionARN}.");
        Console.WriteLine(new string('-', 80));

        return executionARN;
    }

    /// <summary>
    /// Wait for a pipeline run to complete.
    /// </summary>
    /// <param name="executionArn">The pipeline run ARN.</param>
    /// <returns>Async task.</returns>
    public static async Task WaitForPipelineExecution(string executionArn)
    {
```

```

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Waiting for pipeline to finish.");

        PipelineExecutionStatus status;
        do
        {
            status = await
_sageMakerWrapper.CheckPipelineExecutionStatus(executionArn);
            Thread.Sleep(30000);
            Console.WriteLine($"\\tStatus is {status}.");
        } while (status == PipelineExecutionStatus.Executing);

        Console.WriteLine($"\\tPipeline finished with status {status}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="askUser">True to ask the user for cleanup.</param>
    /// <param name="queueUrl">The URL of the queue to clean up.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> CleanupResources(
        bool askUser,
        string queueUrl,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (!askUser || GetYesNoResponse($"\\tDelete pipeline {pipelineName}? (y/
n)"))
        {
            Console.WriteLine($"\\tDeleting pipeline.");
            // Delete the pipeline.
            await _sageMakerWrapper.DeletePipelineByName(pipelineName);
        }

        if (!string.IsNullOrEmpty(queueUrl) && (!askUser ||
GetYesNoResponse($"\\tDelete queue {queueUrl}? (y/n)"))))
        {

```

```
        Console.WriteLine($"\\tDeleting queue.");
        // Delete the queue.
        await _sqsClient.DeleteQueueAsync(new DeleteQueueRequest(queueUrl));
    }

    if (!askUser || GetYesNoResponse($"\\tDelete Amazon S3 bucket {bucketName}?
(y/n)"))
    {
        Console.WriteLine($"\\tDeleting bucket.");
        // Delete all objects in the bucket.
        var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
        {
            BucketName = bucketName
        });
        if (deleteList.KeyCount > 0)
        {
            await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.S3Objects
                    .Select(o => new KeyVersion { Key = o.Key }).ToList()
            });
        }

        // Now delete the bucket.
        await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
        {
            BucketName = bucketName
        });
    }

    if (!askUser || GetYesNoResponse($"\\tDelete lambda {lambdaFunctionName}? (y/
n)"))
    {
        Console.WriteLine($"\\tDeleting lambda function.");

        await _lambdaClient.DeleteFunctionAsync(new DeleteFunctionRequest()
        {
            FunctionName = lambdaFunctionName
        });
    }

    if (!askUser || GetYesNoResponse($"\\tDelete role {lambdaRoleName}? (y/n)"))
```



```
    {
        Console.WriteLine($"\\tDetaching policies and deleting role.");

        foreach (var policy in lambdaRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = lambdaRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = lambdaRoleName
        });
    }

    if (!askUser || GetYesNoResponse($"\\tDelete role {sageMakerRoleName}? (y/
n)"))
    {
        Console.WriteLine($"\\tDetaching policies and deleting role.");

        foreach (var policy in sageMakerRolePolicies)
        {
            await _iamClient!.DetachRolePolicyAsync(new
DetachRolePolicyRequest()
            {
                RoleName = sageMakerRoleName,
                PolicyArn = policy
            });
        }

        await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
        {
            RoleName = sageMakerRoleName
        });
    }

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

```
/// <summary>
/// Helper method to get a role's ARN if it already exists.
/// </summary>
/// <param name="roleName">The name of the AWS Identity and Access Management
(IAM) Role to look for.</param>
/// <returns>The role ARN if it exists, otherwise an empty string.</returns>
private static async Task<string> GetRoleArnIfExists(string roleName)
{
    Console.WriteLine($"Checking for role named {roleName}.");

    try
    {
        var existingRole = await _iamClient.GetRoleAsync(new GetRoleRequest()
        {
            RoleName = roleName
        });
        return existingRole.Role.Arn;
    }
    catch (NoSuchEntityException)
    {
        return string.Empty;
    }
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question)
{
    Console.WriteLine(question);
    var ynResponse = Console.ReadLine();
    var response = ynResponse != null &&
        ynResponse.Equals("y",
            StringComparison.InvariantCultureIgnoreCase);
    return response;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreatePipeline](#)

- [DeletePipeline](#)
- [DescribePipelineExecution](#)
- [StartPipelineExecution](#)
- [UpdatePipeline](#)

使用 AWS SDK for .NET 的 Secrets Manager 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Secrets Manager 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

获取密钥值

以下代码示例演示了如何获取 Secrets Manager 密钥值。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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>    /// <param name="secretName">The name of the secret value to retrieve.</  
    /// <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;
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetSecretValue](#) 中的。

使用 AWS SDK for .NET 的 Amazon SES 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon SES 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

创建电子邮件模板

以下代码示例演示了如何创建 Amazon SES 电子邮件模板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateTemplate](#) 中的。

删除电子邮件模板

以下代码示例演示了如何删除 Amazon SES 电子邮件模板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTemplate](#) 中的。

删除身份

以下代码示例演示了如何删除 Amazon SES 身份。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteIdentity](#) 中的。

获取发送限制

以下代码示例显示如何获取 Amazon SES 发送限制。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetSendQuota](#) 中的。

获取身份的状态

以下代码示例显示如何获取 Amazon SES 身份的状态。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetIdentityVerificationAttributes](#) 中的。

列出电子邮件模板

以下代码示例演示了如何列出 Amazon SES 电子邮件模板。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTemplates](#) 中的。

列出身份

以下代码示例演示了如何列出 Amazon SES 身份。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListIdentities](#) 中的。

发送电子邮件

以下代码示例演示了如何通过 Amazon SES 发送电子邮件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SendEmail](#) 中的。

发送模板化电子邮件

以下代码示例演示了如何通过 Amazon SES 发送模板化电子邮件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SendTemplatedEmail](#) 中的。

验证电子邮件身份

以下代码示例演示了如何通过 Amazon SES 验证电子邮件身份。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [VerifyEmailIdentity](#) 中的。

使用 AWS SDK for .NET 的 Amazon SNS 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 Amazon SNS 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon SNS

以下代码示例演示了如何开始使用 Amazon SNS。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTopics](#) 中的。

主题

- [操作](#)
- [场景](#)
- [无服务器示例](#)

操作

检查电话号码是否选择不接收消息

以下代码示例演示了如何检查电话号码是否退出接收 Amazon SNS 消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CheckIfPhoneNumbersIsOptedOut](#) 中的。

创建主题

以下代码示例演示了如何创建 Amazon SNS 主题。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用特定的名称创建主题。

```
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;
    }
}
```

创建一个包含名称以及特定 FIFO 和重复数据消除属性的新主题。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateTopic](#) 中的。

删除订阅

以下代码示例演示了如何删除 Amazon SNS 订阅。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

通过订阅 ARN 取消订阅某个主题。

```
/// <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;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [Unsubscribe](#)。

删除主题

以下代码示例演示了如何删除 Amazon SNS 主题以及该主题的所有订阅。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

按主题 ARN 删除主题。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTopic](#) 中的。

获取主题属性

以下代码示例演示了如何获取 Amazon SNS 主题的属性。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetTopicAttributes](#) 中的。

列出主题订阅用户

以下代码示例演示了如何检索 Amazon SNS 主题的订阅者列表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
    }
}
```

```
    }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListSubscriptions](#) 中的。

列出主题

以下代码示例演示了如何列出 Amazon SNS 主题。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTopics](#) 中的。

发布带有属性的消息

以下代码示例演示了如何使用 Amazon SNS 发布带有属性的消息。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用组、复制和属性选项向主题发布消息。

```
/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                "\r\nAll messages within the same group will be
received in the order " +
                "they were published.");

            Console.WriteLine();
            var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
```



```

        "you must enter a deduplication ID.");

        Console.WriteLine("Enter a deduplication ID for this message.");
        deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
    }

    if (GetYesNoResponse("Add an attribute to this message?"))
    {
        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageId = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

```

将用户的选择应用于发布操作。

```

/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Publish](#)。

发布 SMS 文本消息

以下代码示例演示了如何使用 Amazon SNS 发布 SMS 消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 [Publish](#)。

发布到主题

以下代码示例演示了如何将消息发布到 Amazon SNS 主题。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

向主题发布消息。

```
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}");
    }
}
```

向主题发布带有属性、可选的重复数据删除和组 ID 的消息。

```
/// <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;
}

```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Publish](#)。

针对主题订阅 SQS 队列

以下代码示例演示了如何订阅 Amazon SQS 队列，以便它接收来自 Amazon SNS 主题的通知。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用可选筛选条件为队列订阅主题。

```

/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Subscribe](#)。

为主题订阅电子邮件地址

以下代码示例演示了如何针对 Amazon SNS 主题订阅电子邮件地址。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Subscribe](#)。

针对主题使用筛选条件进行订阅

以下代码示例演示了如何针对 Amazon SNS 主题使用筛选条件进行订阅。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用可选筛选条件为队列订阅主题。

```
/// <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;
}
```

- 有关 API 详细信息，请参阅 AWS SDK for .NET API 参考中的 [Subscribe](#)。

场景

将消息发布到队列

以下代码示例演示了操作流程：

- 创建主题 (FIFO 或非 FIFO)。
- 针对主题订阅多个队列，并提供应用筛选条件的选项。
- 将消息发布到主题。
- 轮询队列中是否有收到的消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
/// <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\n}and queue URL {queueUrl}" +
                $"{r\n}has been created.\r\n");

            if (i == 0)
            {
                Console.WriteLine(
                    $"The queue URL is used to retrieve the queue ARN,\r\n" +
                    $"which is used to create a subscription.");
                Console.WriteLine(new string('-', 80));
            }

            var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

            if (i == 0)
            {
                Console.WriteLine(
                    $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
```

```
        $"messages from an SNS topic");
    }

    await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

    await SetupFilters(i, queueArn, queueName);
}
}

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

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +
                "If you add a filter to this subscription, then only the
filtered messages " +
                "will be received in the queue.");

            Console.WriteLine(
                "For information about message filtering, " +
                "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

            Console.WriteLine(
                "For this example, you can filter messages by a " +
                "TONE attribute.");
        }
    }
}
```



```
        var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

        string? filterPolicy = null;
        if (useFilter)
        {
            filterPolicy = CreateFilterPolicy();
        }
        var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
            queueArn);
        _subscriptionArns[queueCount] = subscriptionArn;

        Console.WriteLine(
            $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
            $"with the subscription ARN {subscriptionArn}");
        Console.WriteLine(new string('-', 80));
    }
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
        }
    }
}
```

```

    }

    var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
    int.TryParse(selection, out selectionNumber);
    if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
    {
        filterSelections.Add(_tones[selectionNumber - 1]);
    }
} while (selectionNumber != 0);

var filters = new Dictionary<string, List<string>>
{
    { "tone", filterSelections }
};
string filterPolicy = JsonSerializer.Serialize(filters);
return filterPolicy;
}

/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                "\r\nAll messages within the same group will be
received in the order " +
                "they were published.");
        }
    }
}

```

```
        Console.WriteLine();
        var messageId = GetUserResponse("Enter a message ID for
this message:", "1");

        if (!_useContentBasedDeduplication)
        {
            Console.WriteLine("Because you are not using content-based
deduplication, " +
                "you must enter a deduplication ID.");

            Console.WriteLine("Enter a deduplication ID for this message.");
            deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
        }

        if (GetYesNoResponse("Add an attribute to this message?"))
        {
            Console.WriteLine("Enter a number for an attribute.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", "1");
            int.TryParse(selection, out var selectionNumber);

            if (selectionNumber > 0 && selectionNumber < _tones.Length)
            {
                toneAttribute = _tones[selectionNumber - 1];
            }
        }

        var messageId = await SnsWrapper.PublishToTopicWithAttribute(
            _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

        Console.WriteLine($"Message published with id {messageId}.");
    }

    keepSendingMessages = GetYesNoResponse("Send another message?", false);
}
}

/// <summary>
```

```
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();
    while (moreMessages)
    {
        var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

        moreMessages = newMessages.Any();
        if (moreMessages)
        {
            messages.AddRange(newMessages);
        }
    }

    Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

    foreach (var message in messages)
    {
        Console.WriteLine("\tMessage:" +
            $" \n\t{message.Body}");
    }

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

/// <summary>
/// Delete the message using handles in a batch.
/// </summary>
/// <returns>Async task.</returns>
```

```
public static async Task DeleteMessages(string queueUrl, List<Message> messages)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
    await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Clean up the resources from the scenario.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CleanupResources()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    try
    {
        foreach (var queueUrl in _queueUrls)
        {
            if (!string.IsNullOrEmpty(queueUrl))
            {
                var deleteQueue =
                    GetYesNoResponse($"Delete queue with url {queueUrl}?");
                if (deleteQueue)
                {
                    await SqsWrapper.DeleteQueueByUrl(queueUrl);
                }
            }
        }

        foreach (var subscriptionArn in _subscriptionArns)
        {
            if (!string.IsNullOrEmpty(subscriptionArn))
            {
                await SnsWrapper.UnsubscribeByArn(subscriptionArn);
            }
        }

        var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
        if (deleteTopic)
        {
```

```
        await SnsWrapper.DeleteTopicByArn(_topicArn);
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
}

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

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
{
    if (UseConsole)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);

        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}

/// <summary>
/// Helper method to get a string response from the user through the console.
/// </summary>
/// <param name="question">The question string to print on the console.</param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static string GetUserResponse(string question, string defaultAnswer)
{
    if (UseConsole)
    {
        var response = "";
    }
}
```

```

        while (string.IsNullOrEmpty(response))
        {
            Console.WriteLine(question);
            response = Console.ReadLine();
        }
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}
}

```

创建一个包装 Amazon SQS 操作的类。

```

/// <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;
}
}

```

创建一个包装 Amazon SNS 操作的类。

```

/// <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;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [订阅](#)
 - [Unsubscribe](#)

无服务器示例

通过 Amazon SNS 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收因接收来自 SNS 主题的消息而触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

AWS SDK for .NET

 Note

还有更多相关信息 [GitHub](#)。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

使用 .NET 将 SNS 事件与 Lambda 结合使用。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
    }
}
```



```
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
```

使用 AWS SDK for .NET 的 Amazon SQS 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon SQS 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Amazon SQS

以下代码示例演示了如何开始使用 Amazon SQS。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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();
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListQueues](#) 中的。

主题

- [操作](#)
- [场景](#)
- [无服务器示例](#)

操作

授权桶向队列发送消息

以下代码示例演示如何授权 Amazon S3 桶向 Amazon SQS 队列发送消息。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;

public class AuthorizeS3ToSendMessage
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// AuthorizeS3ToSendMessageAsync method to authorize the named
    /// bucket to send messages in response to S3 events.
    /// </summary>
    public static async Task Main()
    {
        string queueUrl = "https://sqs.us-east-2.amazonaws.com/0123456789ab/
Example_Queue";
        string bucketName = "doc-example-bucket";

        // Create an Amazon SQS client object using the
        // default user. If the AWS Region you want to use
        // is different, supply the AWS Region as a parameter.
        IAmazonSQS client = new AmazonSQSClient();

        var queueARN = await client.AuthorizeS3ToSendMessageAsync(queueUrl,
bucketName);

        if (!string.IsNullOrEmpty(queueARN))
        {
            Console.WriteLine($"The Amazon S3 bucket: {bucketName} has been
successfully authorized.");
            Console.WriteLine($"{bucketName} can now send messages to the queue
with ARN: {queueARN}.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SendMessage](#) 中的。

创建队列

以下代码示例演示了如何创建 Amazon SQS 队列。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用特定的名称创建队列。

```
/// <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;
}
```

创建 Amazon SQS 队列并向其发送消息。

```
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;
    }
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateQueue](#) 中的。

从队列中删除一批消息

以下代码示例演示了如何从 Amazon SQS 队列中删除一批消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteMessageBatch](#) 中的。

从队列删除消息

以下代码示例演示了如何从 Amazon SQS 队列中删除消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

接收来自 Amazon SQS 队列的消息，然后删除该消息。

```
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;
}
```

```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteMessage](#) 中的。

删除队列

以下代码示例演示了如何删除 Amazon SQS 队列。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用此 URL 删除队列。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteQueue](#) 中的。

获取队列的属性

以下代码示例演示了如何获取 Amazon SQS 队列的属性。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetQueueAttributes](#) 中的。

获取队列的 URL

以下代码示例演示了如何获取 Amazon SQS 队列的 URL。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetQueueUrl](#) 中的。

接收来自队列的消息

以下代码示例演示了如何接收来自 Amazon SQS 队列的消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用队列的 URL 接收来自队列的消息。

```
/// <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;
}
```

接收来自 Amazon SQS 队列的消息，然后删除该消息。

```
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;
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ReceiveMessage](#) 中的。

向队列发送消息

以下代码示例演示了如何向 Amazon SQS 队列发送消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建 Amazon SQS 队列并向其发送消息。

```
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;
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SendMessage](#) 中的。

设置队列属性

以下代码示例演示了如何设置 Amazon SQS 队列的属性。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

为主题设置队列的策略属性。

```
/// <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;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SetQueueAttributes](#) 中的。

场景

将消息发布到队列

以下代码示例演示了操作流程：

- 创建主题（FIFO 或非 FIFO）。
- 针对主题订阅多个队列，并提供应用筛选条件的选项。

- 将消息发布到主题。
- 轮询队列中是否有收到的消息。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
/// <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($"{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;
    }
}
```

创建一个包装 Amazon SQS 操作的类。

```
/// <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;
}
}
```

创建一个包装 Amazon SNS 操作的类。

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSNS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
    attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
    duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
    useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
```

```
        { "FifoTopic", "true" }
    };
    if (useContentBasedDeduplication)
    {
        createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
    }
}

var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
return createResponse.TopicArn;
}

/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }

    var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}

/// <summary>
```

```
    /// Publish a message to a topic with an attribute and optional deduplication
    and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String" } }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
}
```



```
/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)

- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [订阅](#)
- [Unsubscribe](#)

无服务器示例

通过 Amazon SQS 触发器调用 Lambda 函数

以下代码示例演示如何实现一个 Lambda 函数，该函数接收从 SQS 队列收到消息时触发的事件。该函数从事件参数检索消息并记录每条消息的内容。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在[无服务器示例](#)存储库中查找完整示例，并了解如何进行设置和运行。

通过 .NET 将 SQS 事件与 Lambda 结合使用。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }
    }
}
```

```
        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

报告使用 Amazon SQS 触发器进行 Lambda 函数批处理项目失败

以下代码示例演示如何为从 SQS 队列接收事件的 Lambda 函数实现部分批处理响应。该函数在响应中报告批处理项目失败，并指示 Lambda 稍后重试这些消息。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [无服务器示例](#) 存储库中查找完整示例，并了解如何进行设置和运行。

报告使用 .NET 进行 Lambda SQS 批处理项目失败。

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

使用 AWS SDK for .NET 的 Step Functions 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Step Functions 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 Step Functions

以下代码示例演示如何开始使用 Step Functions。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListStateMachines](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

创建状态机

以下代码示例演示如何创建 Step Functions 状态机。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateStateMachine](#) 中的。

创建活动

以下代码示例演示如何创建 Step Functions 活动。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateActivity](#) 中的。

删除状态机

以下代码示例演示如何删除 Step Functions 状态机。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteStateMachine](#) 中的。

删除活动

以下代码示例演示如何删除 Step Functions 活动。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteActivity](#) 中的。

描述状态机

以下代码示例演示如何描述 Step Functions 状态机。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeStateMachine](#) 中的。

描述状态机运行

以下代码示例演示如何描述 Step Functions 状态机运行情况。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeExecution](#) 中的。

获取活动的任务数据

以下代码示例演示如何获取 Step Functions 活动的任务数据。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }

```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetActivityTask](#) 中的。

列出活动

以下代码示例演示如何列出 Step Functions 活动。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```

    /// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListActivities](#) 中的。

列出状态机运行情况

以下代码示例演示如何列出 Step Functions 状态机运行情况。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListExecutions](#) 中的。

列出状态机

以下代码示例演示如何列出 Step Functions 状态机。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListStateMachines](#) 中的。

向任务发送成功响应

以下代码示例演示如何向 Step Functions 任务发送成功响应。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [SendTaskSuccess](#) 中的。

启动状态机运行

以下代码示例演示如何启动 Step Functions 状态机运行。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```


- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartExecution](#) 中的。

场景

开始使用状态机

以下代码示例演示了操作流程：

- 创建活动。
- 根据包含先前创建的活动作为步骤的 Amazon States Language 定义创建状态机。
- 运行状态机并使用用户输入响应活动。
- 运行完成后获取最终状态和输出，然后清理资源。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
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);
}
}

```

定义一个包装状态机和活动操作的类。

```

namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

/// <summary>
/// Wrapper that performs AWS Step Functions actions.
/// </summary>
public class StepFunctionsWrapper
{
    private readonly IAmazonStepFunctions _amazonStepFunctions;

    /// <summary>
    /// The constructor for the StepFunctionsWrapper. Initializes the
    /// client object passed to it.
    /// </summary>
    /// <param name="amazonStepFunctions">An initialized Step Functions client
    object.</param>

```

```
public StepFunctionsWrapper(IAmazonStepFunctions amazonStepFunctions)
{
    _amazonStepFunctions = amazonStepFunctions;
}

/// <summary>
/// Create a Step Functions activity using the supplied name.
/// </summary>
/// <param name="activityName">The name for the new Step Functions activity.</
param>
/// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
public async Task<string> CreateActivity(string activityName)
{
    var response = await _amazonStepFunctions.CreateActivityAsync(new
CreateActivityRequest { Name = activityName });
    return response.ActivityArn;
}

/// <summary>
/// Create a Step Functions state machine.
/// </summary>
/// <param name="stateMachineName">Name for the new Step Functions state
/// machine.</param>
/// <param name="definition">A JSON string that defines the Step Functions
/// state machine.</param>
/// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
/// <returns></returns>
public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
{
    var request = new CreateStateMachineRequest
    {
        Name = stateMachineName,
        Definition = definition,
        RoleArn = roleArn
    };

    var response =
        await _amazonStepFunctions.CreateStateMachineAsync(request);
    return response.StateMachineArn;
}
```

```
/// <summary>
/// Delete a Step Machine activity.
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the activity.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteActivity(string activityArn)
{
    var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Delete a Step Functions state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// state machine.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteStateMachine(string stateMachineArn)
{
    var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
    { StateMachineArn = stateMachineArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Retrieve information about the specified Step Functions execution.
/// </summary>
/// <param name="executionArn">The Amazon Resource Name (ARN) of the
/// Step Functions execution.</param>
/// <returns>The API response returned by the API.</returns>
public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
{
    var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
    return response;
}
```

```
/// <summary>
/// Retrieve information about the specified Step Functions state machine.
/// </summary>
/// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine to retrieve.</param>
/// <returns>Information about the specified Step Functions state machine.</
returns>
public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
StateMachineArn)
{
    var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
    return response;
}

/// <summary>
/// Retrieve a task with the specified Step Functions activity
/// with the specified Amazon Resource Name (ARN).
/// </summary>
/// <param name="activityArn">The Amazon Resource Name (ARN) of
/// the Step Functions activity.</param>
/// <param name="workerName">The name of the Step Functions worker.</param>
/// <returns>The response from the Step Functions activity.</returns>
public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
{
    var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
    { ActivityArn = activityArn, WorkerName = workerName });
    return response;
}

/// <summary>
/// List the Step Functions activities for the current account.
/// </summary>
/// <returns>A list of ActivityListItem.</returns>
public async Task<List<ActivityListItem>> ListActivitiesAsync()
{
    var request = new ListActivitiesRequest();
    var activities = new List<ActivityListItem>();

    do
```

```
    {
        var response = await _amazonStepFunctions.ListActivitiesAsync(request);

        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }

        activities.AddRange(response.Activities);
    }
    while (request.NextToken is not null);

    return activities;
}

/// <summary>
/// Retrieve information about executions of a Step Functions
/// state machine.
/// </summary>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>A list of ExecutionListItem objects.</returns>
public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
{
    var executions = new List<ExecutionListItem>();
    ListExecutionsResponse response;
    var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

    do
    {
        response = await _amazonStepFunctions.ListExecutionsAsync(request);
        executions.AddRange(response.Executions);
        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }
    } while (response.NextToken is not null);

    return executions;
}
```

```
/// <summary>
/// Retrieve a list of Step Functions state machines.
/// </summary>
/// <returns>A list of StateMachineListItem objects.</returns>
public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
{
    var stateMachines = new List<StateMachineListItem>();
    var listStateMachinesPaginator =
        _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

    await foreach (var response in listStateMachinesPaginator.Responses)
    {
        stateMachines.AddRange(response.StateMachines);
    }

    return stateMachines;
}

/// <summary>
/// Indicate that the Step Functions task, indicated by the
/// task token, has completed successfully.
/// </summary>
/// <param name="taskToken">Identifies the task.</param>
/// <param name="taskResponse">The response received from executing the task.</
param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> SendTaskSuccessAsync(string taskToken, string
taskResponse)
{
    var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
SendTaskSuccessRequest
    { TaskToken = taskToken, Output = taskResponse });

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Start execution of an AWS Step Functions state machine.
/// </summary>
/// <param name="executionName">The name to use for the execution.</param>
```

```
/// <param name="executionJson">The JSON string to pass for execution.</param>
/// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
/// Step Functions state machine.</param>
/// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
/// execution.</returns>
public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
{
    var executionRequest = new StartExecutionRequest
    {
        Input = executionJson,
        StateMachineArn = stateMachineArn
    };

    var response = await
_amazonStepFunctions.StartExecutionAsync(executionRequest);
    return response.ExecutionArn;
}

/// <summary>
/// Stop execution of a Step Functions workflow.
/// </summary>
/// <param name="executionArn">The Amazon Resource Name (ARN) of
/// the Step Functions execution to stop.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StopExecution(string executionArn)
{
    var response =
        await _amazonStepFunctions.StopExecutionAsync(new StopExecutionRequest
{ ExecutionArn = executionArn });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [CreateActivity](#)
 - [CreateStateMachine](#)

- [DeleteActivity](#)
- [DeleteStateMachine](#)
- [DescribeExecution](#)
- [DescribeStateMachine](#)
- [GetActivityTask](#)
- [ListActivities](#)
- [ListStateMachines](#)
- [SendTaskSuccess](#)
- [StartExecution](#)
- [StopExecution](#)

使用 AWS SDK for .NET 的 AWS STS 示例

以下代码示例演示了如何将AWS SDK for .NET 与 AWS STS 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

代入角色

以下代码示例演示了如何通过 AWS STS 代入角色。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    }
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AssumeRole](#) 中的。

使用 AWS SDK for .NET 的 AWS Support 示例

以下代码示例演示了如何将 AWS SDK for .NET 与 AWS Support 结合使用，以执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

开始使用

开始使用 AWS Support

以下代码示例演示了如何开始使用 AWS Support。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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.");
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeServices](#) 中的。

主题

- [操作](#)
- [场景](#)

操作

向案例添加通信

以下代码示例演示了如何将带附件的 AWS Support 通信添加到支持案例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AddCommunicationToCase](#) 中的。

向集合添加附件

以下代码示例演示了如何将 AWS Support 附件添加到附件集。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [AddAttachmentsToSet](#) 中的。

创建案例

以下代码示例演示了如何创建新的 AWS Support 案例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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 "en" (English) and "ja" (Japanese) 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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateCase](#) 中的。

描述附件

以下代码示例演示了如何描述 AWS Support 案例的附件。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeAttachment](#) 中的。

描述案例

以下代码示例演示了如何描述 AWS Support 案例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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 "en" (English) and "ja" (Japanese) 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;  
    }  
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeCases](#) 中的。

描述通信

以下代码示例演示了如何描述案例的 AWS Support 通信。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeCommunications](#) 中的。

描述服务

以下代码示例演示如何描述 AWS 服务列表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <summary>
/// Get the descriptions of AWS services.
/// </summary>
/// <param name="name">Optional language for services.
/// Currently "en" (English) and "ja" (Japanese) 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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeServices](#) 中的。

描述严重性级别

以下代码示例演示了如何描述 AWS Support 严重性级别。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <summary>
/// Get the descriptions of support severity levels.
/// </summary>
/// <param name="name">Optional language for severity levels.
/// Currently "en" (English) and "ja" (Japanese) 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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeSeverityLevels](#) 中的。

解析案例

以下代码示例演示了如何解析 AWS Support 案例。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ResolveCase](#) 中的。

场景

开始应用场景

以下代码示例演示了操作流程：

- 获取并显示案例的可用服务和严重级别。
- 使用选定的服务、类别和严重性级别创建支持案例。
- 获取并显示当天打开案例的列表。
- 向新案例添加附件集和通信。
- 描述该案例的新附件和通信。
- 解析案例。
- 获取并显示当天未解决的案例列表。

AWS SDK for .NET

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
/// <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($"2. Available support categories for Service
\"{service.Name}\":");
    for (int i = 0; i < service.Categories.Count; i++)
    {
        Console.WriteLine($"  {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));
}
}
```

场景用于 AWS Support 操作的包装程序方法。

```
/// <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 "en" (English) and "ja" (Japanese) 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 "en" (English) and "ja" (Japanese) 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 "en" (English) and "ja" (Japanese) 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 "en" (English) and "ja" (Japanese) 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;
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)

- [DescribeCases](#)
- [DescribeCommunications](#)
- [DescribeServices](#)
- [DescribeSeverityLevels](#)
- [ResolveCase](#)

使用 AWS SDK for .NET 的 Amazon Transcribe 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Transcribe 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

创建自定义词汇表

以下代码示例演示如何创建自定义 Amazon Transcribe 词汇表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [CreateVocabulary](#) 中的。

删除自定义词汇表

以下代码示例演示如何删除自定义 Amazon Transcribe 词汇表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteVocabulary](#) 中的。

删除医疗转录任务

以下代码示例演示如何删除 Amazon Transcribe Medical 转录任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteMedicalTranscriptionJob](#) 中的。

删除转录任务

以下代码示例演示如何删除 Amazon Transcribe 转录任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DeleteTranscriptionJob](#) 中的。

获取自定义词汇表

以下代码示例演示如何获取自定义 Amazon Transcribe 词汇表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetVocabulary](#) 中的。

获取转录任务

以下代码示例演示如何获取 Amazon Transcribe 转录任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。


```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [GetTranscriptionJob](#) 中的。

列出自定义词汇表

以下代码示例演示如何列出自定义 Amazon Transcribe 词汇表。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListVocabularies](#) 中的。

列出医疗转录任务

以下代码示例演示如何列出 Amazon Transcribe Medical 转录任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListMedicalTranscriptionJobs](#) 中的。

列出转录任务

以下代码示例演示如何列出 Amazon Transcribe 转录任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTranscriptionJobs](#) 中的。

启动医疗转录任务

以下代码示例演示如何启动 Amazon Transcribe Medical 转录任务。

AWS SDK for .NET

Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartMedicalTranscriptionJob](#) 中的。

启动转录任务

以下代码示例演示如何启动 Amazon Transcribe 转录任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```



```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartTranscriptionJob](#) 中的。

更新自定义词汇表

以下代码示例演示如何更新自定义 Amazon Transcribe 词汇表。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
/// <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;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [UpdateVocabulary](#) 中的。

使用 AWS SDK for .NET 的 Amazon Translate 示例

以下代码示例演示如何将 AWS SDK for .NET 与 Amazon Translate 结合使用来执行操作和实现常见场景。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是指显示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何在上下文中设置和运行代码的说明。

主题

- [操作](#)

操作

描述翻译任务

以下代码示例演示如何描述 Amazon Translate 翻译任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [DescribeTextTranslationJob](#) 中的。

列出翻译任务

以下代码示例演示如何列出 Amazon Translate 翻译任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
    });
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListTextTranslationJobs](#) 中的。

启动翻译任务

以下代码示例演示如何启动 Amazon Translate 翻译任务。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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;
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StartTextTranslationJob](#) 中的。

停止翻译任务

以下代码示例演示如何停止 Amazon Translate 翻译任务。

AWS SDK for .NET

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [StopTextTranslationJob](#) 中的。

翻译文本

以下代码示例演示如何使用 Amazon Translate 翻译文本。

AWS SDK for .NET

Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
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 the language of the source text.

// You can get a list of the languages supported 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);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [TranslateText](#) 中的。

使用 AWS SDK for .NET 的跨服务示例

以下示例应用程序使用 AWS SDK for .NET 来跨多个 AWS 服务工作。

跨服务示例以高级经验为目标，可帮助您开始构建应用程序。

示例

- [构建转换消息的发布和订阅应用程序](#)
- [创建照片资产管理应用程序，让用户能够使用标签管理照片](#)
- [创建 Web 应用程序来跟踪 DynamoDB 数据](#)
- [创建 Aurora Serverless 工作项跟踪器](#)
- [创建用于分析客户反馈和合成音频的应用程序](#)
- [使用 AWS SDK 通过 Amazon Rekognition 检测图像中的对象](#)

构建转换消息的发布和订阅应用程序

AWS SDK for .NET

展示如何使用 Amazon Simple Notification Service .NET API 创建具有订阅和发布功能的 Web 应用程序。此外，此示例应用程序还会转换消息。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 [GitHub](#)。

本示例中使用的服务

- Amazon SNS
- Amazon Translate

创建照片资产管理应用程序，让用户能够使用标签管理照片

AWS SDK for .NET

演示了如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 [GitHub](#)。

要深入了解这个例子的起源，请参阅 [AWS 社区](#) 上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

创建 Web 应用程序来跟踪 DynamoDB 数据

AWS SDK for .NET

展示如何使用 Amazon DynamoDB .NET API 创建用于跟踪 DynamoDB 工作数据的动态 Web 应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 [GitHub](#)。

本示例中使用的服务

- DynamoDB
- Amazon SES

创建 Aurora Serverless 工作项跟踪器

AWS SDK for .NET

展示如何使用 AWS SDK for .NET 通过 Amazon Simple Email Service (Amazon SES) 创建 Web 应用程序来跟踪 Amazon Aurora 数据库中的工作项和电子邮件报告。此示例使用由 React.js 构建的前端与 RESTful .NET 后端进行交互。

- 将 React Web 应用程序与 AWS 服务集成。
- 列出、添加、更新和删除 Aurora 表中的项目。
- 使用 Amazon SES 以电子邮件形式发送已筛选工作项的报告。
- 使用随附的 AWS CloudFormation 脚本部署与管理示例资源。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例[GitHub](#)。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

创建用于分析客户反馈和合成音频的应用程序

AWS SDK for .NET

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。
- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 AWS CDK 进行部署。有关源代码和部署说明，请参阅中的项目 [GitHub](#)。

本示例中使用的服务

- Amazon Comprehend

- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

使用 AWS SDK 通过 Amazon Rekognition 检测图像中的对象

AWS SDK for .NET

展示如何使用 Amazon Rekognition .NET API 创建应用程序，该应用程序采用 Amazon Rekognition 来按类别识别位于 Amazon Simple Storage Service (Amazon S3) 存储桶的图像中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例[GitHub](#)。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

此 AWS 产品或服务的安全性

云安全性一直是 Amazon Web Services (AWS) 的重中之重。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。安全性是 AWS 和您的共同责任。[责任共担模式](#)将其描述为云的安全性和云中的安全性。

云的安全性 – AWS 负责保护运行在 AWS 云中提供的所有服务的基础设施，并向您提供可安全使用的服务。安全责任是 AWS 的最高优先任务，作为 [AWS 合规性计划](#) 的一部分，我们安全性的有效性由第三方审计员定期进行测试和验证。

云中的安全性 – 您的责任由您所使用的 AWS 服务以及其他因素决定，包括您数据的敏感性、您企业的要求以及适用的法律法规。

此 AWS 产品或服务通过它支持的特定 Amazon Web Services (AWS) 服务遵循[责任共担模式](#)。有关 AWS 服务安全性信息，请参阅 [AWS 服务安全性文档页面](#) 和 [合规性计划规定的 AWS 合规性工作范围内的 AWS 服务](#)。

主题

- [本 AWS 产品或服务中的数据保护](#)
- [Identity and Access Management](#)
- [此 AWS 产品或服务的合规性验证](#)
- [此 AWS 产品或服务的故障恢复能力](#)
- [此 AWS 产品或服务的基础设施安全性](#)
- [强制在 AWS SDK for .NET 中实施最低 TLS 版本](#)
- [Amazon S3 加密客户端迁移](#)

本 AWS 产品或服务中的数据保护

AWS [责任共担模式](#) 适用于本 AWS 产品或服务中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础架构上的内容的控制。您还负责您所使用的 AWS 服务的安全配置和管理任务。有关数据隐私的更多信息，请参阅[数据隐私常见问题解答](#)。有关欧洲数据保护的信息，请参阅 AWS 安全性博客上的 [AWS 责任共担模式和 GDPR](#) 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与AWS资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用AWS CloudTrail设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务 (例如 Amazon Macie)，它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果在通过命令行界面或 API 访问AWS时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅 [《美国联邦信息处理标准 \(FIPS \) 第 140-2 版》](#)。

我们强烈建议您切勿将机密信息或敏感信息 (如您客户的电子邮件地址) 放入标签或自由格式文本字段 (如名称字段)。这包括使用本 AWS 产品或服务或者其他使用控制台、API、AWS CLI 或 AWS SDK 的 AWS 服务时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

Identity and Access Management

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证 (登录) 和授权 (具有权限) 来使用 AWS 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

- [受众](#)
- [使用身份进行身份验证](#)
- [使用策略管理访问](#)
- [AWS 服务如何与 IAM 协同工作](#)
- [对 AWS 身份和访问进行问题排查](#)

受众

使用 AWS Identity and Access Management (IAM) 的方式因您可以在 AWS 中执行的操作而异。

服务用户 – 如果使用 AWS 服务来执行任务，则管理员会为您提供所需的凭证和权限。当您使用更多 AWS 特征来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的

权限。如果您无法访问 AWS 中的功能，请参阅[对 AWS 身份和访问进行问题排查](#)或您所使用的 AWS 服务的用户指南。

服务管理员 - 如果您在公司负责管理 AWS 资源，则您可能具有 AWS 的完全访问权限。您有责任确定您的服务用户应访问哪些 AWS 特征和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何将 IAM 与 AWS 结合使用，请参阅您所使用的 AWS 服务的用户指南。

IAM 管理员 - 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 AWS 的访问权限的详细信息。要查看您可在 IAM 中使用的 AWS 基于身份的策略示例，请参阅您所使用的 AWS 服务的用户指南。

使用身份进行身份验证

身份验证是使用身份凭证登录AWS的方法。您必须作为AWS 账户根用户、IAM 用户或通过代入 IAM 角色进行身份验证（登录到AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到AWS。AWS IAM Identity Center(IAM Identity Center) 用户、公司的单点登录身份验证以及 Google 或 Facebook 凭证都是联合身份的示例。当以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当使用联合身份验证访问AWS时，就是在间接代入角色。

根据用户类型，可以登录AWS Management Console或AWS访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 用户指南》中的[如何登录到您的 AWS 账户](#)。

如果以编程方式访问AWS，则AWS将提供软件开发工具包（SDK）和命令行界面（CLI），以便使用凭证以加密方式签署请求。如果不使用AWS工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的[签署 AWS API 请求](#)。

无论使用何种身份验证方法，您可能都需要提供其他安全信息。例如，AWS建议使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的[多重身份验证](#)和《IAM 用户指南》中的[在 AWS 中使用多重身份验证（MFA）](#)。

AWS 账户 根用户

创建AWS 账户时，最初使用的是一个对账户中所有AWS 服务和资源拥有完全访问权限的登录身份。此身份称为AWS 账户根用户，使用创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的[需要根用户凭证的任务](#)。

联合身份

作为最佳实操，要求人类用户（包括需要管理员访问权限的用户）结合使用联合身份验证和身份提供程序，以使用临时凭证来访问 AWS 服务。

联合身份是来自企业用户目录、Web 身份提供程序、AWS Directory Service、Identity Center 目录的用户，或任何使用通过身份来源提供的凭证来访问 AWS 服务的用户。当联合身份访问 AWS 账户时，他们担任角色，而角色提供临时凭证。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和组，也可以连接并同步到自己的身份源中的一组用户和组以跨所有 AWS 账户 和应用程序使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的[什么是 IAM Identity Center ?](#)

IAM 用户和组

[IAM 用户](#) 是 AWS 账户内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的[对于需要长期凭证的使用场景定期轮换访问密钥](#)。

[IAM 组](#) 是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。可以使用群组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，可能具有一个名为 IAMAdmins 的群组，并为该群组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的[何时创建 IAM 用户（而不是角色）](#)。

IAM 角色

[IAM 角色](#) 是 AWS 账户中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。可以通过[切换角色](#)，在 AWS Management Console 中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以担任角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的[使用 IAM 角色](#)。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 - 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅

《IAM 用户指南》中的[为第三方身份提供商创建角色](#)。如果使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的[权限集](#)。

- 临时 IAM 用户权限 - IAM 用户或角色可担任 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 - 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些 AWS 服务，可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南 中的 [IAM 角色与基于资源的策略有何不同](#)。
- 跨服务访问 – 某些 AWS 服务使用其他 AWS 服务中的特征。例如，在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service（Amazon S3）中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
 - 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅[转发访问会话](#)。
 - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 [IAM 角色](#)。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的[创建向 AWS 服务委派权限的角色](#)。
 - 服务相关角色 – 服务相关角色是与 AWS 服务关联的一种服务角色。服务可以担任代表您执行操作的角色。服务相关角色显示在 AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon EC2 上运行的应用程序 – 可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的[使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限](#)。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的[何时创建 IAM 角色（而不是用户）](#)。

使用策略管理访问

将创建策略并将其附加到 AWS 身份或资源，以控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，AWS 将评

估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在AWS中存储为JSON文档。有关JSON策略文档的结构和内容的更多信息，请参阅《IAM用户指南》中的[JSON策略概览](#)。

管理员可以使用AWS JSON策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM管理员可以创建IAM策略。然后，管理员可以向角色添加IAM策略，并且用户可以代入角色。

IAM策略定义操作的权限，无关于使用哪种方法执行操作。例如，假设有一个允许iam:GetRole操作的策略。具有该策略的用户可以从AWS Management Console、AWS CLI或AWS API获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如IAM用户、用户组或角色）的JSON权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM用户指南》中的[创建IAM策略](#)。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管式策略是可以附加到AWS账户中的多个用户、组和角色的独立策略。托管式策略包括AWS托管式策略和客户托管式策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM用户指南》中的[在托管式策略与内联策略之间进行选择](#)。

基于资源的策略

基于资源的策略是附加到资源的JSON策略文档。基于资源的策略的示例包括IAM角色信任策略和Amazon S3存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。必须在基于资源的策略中[指定主体](#)。主体可以包括账户、用户、角色、联合用户或AWS服务。

基于资源的策略是位于该服务中的内联策略。不能在基于资源的策略中使用来自IAM的AWS托管式策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL与基于资源的策略类似，尽管它们不使用JSON策略文档格式。

Amazon S3、AWS WAF和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的[访问控制列表 \(ACL\) 概览](#)。

其他策略类型

AWS支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型所授予的最大权限。

- 权限边界 – 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体 (IAM 用户或角色) 授予的最大权限。可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 Principal 字段中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的[IAM 实体的权限边界](#)。
- 服务控制策略 (SCP) - SCP 是 JSON 策略，指定了组织或组织单位 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 服务可以分组和集中管理您的企业拥有的多个 AWS 账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体 (包括每个AWS 账户根用户) 的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的[SCP 的工作原理](#)。
- 会话策略 - 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的[会话策略](#)。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅《IAM 用户指南》中的[策略评估逻辑](#)。

AWS 服务如何与 IAM 协同工作

要大致了解 AWS 服务如何与大多数 IAM 功能结合使用，请参阅《IAM 用户指南》中的[与 IAM 结合使用的 AWS 服务](#)。

要学习如何将特定的 AWS 服务与 IAM 结合使用，请参阅相关服务的《用户指南》的安全部分。

对 AWS 身份和访问进行问题排查

使用以下信息可帮助您诊断和修复在使用 AWS 和 IAM 时可能遇到的常见问题。

主题

- [我无权在 AWS 中执行操作](#)
- [我无权执行 iam : PassRole](#)
- [我希望允许我的 AWS 账户以外的人访问我的 AWS 资源](#)

我无权在 AWS 中执行操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 `aws:GetWidget` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 `aws:GetWidget` 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam : PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 AWS。

有些 AWS 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 AWS 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我希望允许我的 AWS 账户以外的人访问我的 AWS 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACL) 的服务，可以使用这些策略向人员授予对资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 AWS 是否支持这些特征，请参阅 [AWS 服务如何与 IAM 协同工作](#)。
- 要了解如何为您拥有的 AWS 账户 中的资源提供访问权限，请参阅 IAM 用户指南 中的 [为您拥有的另一个 AWS 账户 中的 IAM 用户提供访问权限](#)。
- 要了解如何为第三方 AWS 账户提供您资源的访问权限，请参阅《IAM 用户指南》中的 [为第三方拥有的 AWS 账户提供访问权限](#)。
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的 [为经过外部身份验证的用户 \(身份联合验证\) 提供访问权限](#)。
- 要了解使用角色和基于资源的策略进行跨账户存取之间的差别，请参阅 IAM 用户指南中的 [IAM 角色与基于资源的策略有何不同](#)。

此 AWS 产品或服务的合规性验证

要了解某个 AWS 服务 是否在特定合规性计划范围内，请参阅[合规性计划范围内的 AWS 服务](#)，然后选择您感兴趣的合规性计划。有关常规信息，请参阅 [AWS 合规性计划](#)。

您可以使用 AWS Artifact 下载第三方审核报告。有关更多信息，请参阅[在 AWS Artifact 中下载报告](#)。

您使用 AWS 服务 的合规性责任取决于数据的敏感性、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- [安全性与合规性快速入门指南](#) - 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署以安全性和合规性为重点的基准环境的步骤。
- [Amazon Web Services 上的 HIPAA 安全性和合规性架构设计](#) - 该白皮书介绍了公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。

Note

并非所有 AWS 服务 都符合 HIPAA 要求。有关更多信息，请参阅[符合 HIPAA 要求的服务参考](#)。

- [AWS 合规性资源](#) - 此业务手册和指南集合可能适用于您的行业和位置。
- [AWS 客户合规指南](#)：从合规角度了解责任共担模式。这些指南总结了保护 AWS 服务的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)。
- AWS Config 开发人员指南中的[使用规则评估资源](#) - 此 AWS Config 服务评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- [AWS Security Hub](#) - 此 AWS 服务 向您提供 AWS 中安全状态的全面视图。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实操。有关受支持服务及控制的列表，请参阅 [Security Hub 控制参考](#)。
- [AWS Audit Manager](#) - 此 AWS 服务 可帮助您持续审核您的 AWS 使用情况，以简化管理风险以及与相关法规和行业标准的合规性的方式。

此 AWS 产品或服务通过它支持的特定 Amazon Web Services (AWS) 服务遵循[责任共担模式](#)。有关 AWS 服务安全性信息，请参阅 [AWS 服务安全性文档页面](#)和[合规性计划规定的 AWS 合规性工作范围内的 AWS 服务](#)。

此 AWS 产品或服务的故障恢复能力

AWS 全球基础设施围绕 AWS 区域和可用区而构建。

AWS 区域 提供多个在物理上独立且隔离的可用区，这些可用区与延迟率低、吞吐量高且冗余性高的网络连接在一起。

利用可用区，您可以设计和运营在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅 [AWS全球基础设施](#)。

此 AWS 产品或服务通过它支持的特定 Amazon Web Services (AWS) 服务遵循[责任共担模式](#)。有关 AWS 服务安全性信息，请参阅 [AWS 服务安全性文档页面](#)和[合规性计划规定的 AWS 合规性工作范围内的 AWS 服务](#)。

此 AWS 产品或服务的基础设施安全性

此 AWS 产品或服务使用托管式服务，因此受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础设施的信息，请参阅 [AWS 云安全](#)。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的[基础设施保护](#)。

您可以使用 AWS 发布的 API 调用通过网络访问此 AWS 产品或服务。客户端必须支持以下内容：

- 传输层安全性协议 (TLS)。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE (Ephemeral Diffie-Hellman) 或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。大多数现代系统 (如 Java 7 及更高版本) 都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 生成临时安全凭证来对请求进行签名。

此 AWS 产品或服务通过它支持的特定 Amazon Web Services (AWS) 服务遵循[责任共担模式](#)。有关 AWS 服务安全性信息，请参阅 [AWS 服务安全性文档页面](#)和[合规性计划规定的 AWS 合规性工作范围内的 AWS 服务](#)。

强制在AWS SDK for .NET中实施最低 TLS 版本

要提高与 AWS 服务通信时的安全性，您应将AWS SDK for .NET配置为使用 TLS 1.2 或更高版本。

AWS SDK for .NET 使用底层 .NET 运行时来确定要使用哪个安全协议。默认情况下，当前版本的 .NET 使用操作系统支持的最新的已配置协议。您的应用程序可以覆盖此开发工具包行为，但不建议这样做。

.NET 内核

默认情况下，.NET Core 使用操作系统支持的最新的已配置协议。AWS SDK for .NET 不提供覆盖它的机制。

如果您使用的 .NET Core 版本低于 2.1，我们强烈建议您升级 .NET Core 版本。

有关特定于每个操作系统的信息，请参阅以下内容。

Windows

Windows 的最新发行版[默认情况下启用了](#) TLS 1.2 支持。如果你在 Windows 7 SP1 或 Windows Server 2008 R2 SP1 上运行，则需要确保注册表中启用 TLS 1.2 支持，如 <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings> #tls-12 中所述。如果您正在运行较早的发行版，则必须升级操作系统。有关 Windows 中 TLS 1.3 支持的信息，请查看最新的 Microsoft 文档，了解所需的最低客户端或服务器版本。

macOS

如果您正在运行 .NET Core 2.1 或更高版本，则默认情况下启用 TLS 1.2。[OS X Mavericks v10.9 或更高版本](#)支持 TLS 1.2。[.NET Core 2.1 及更高版本需要更新版本的 macOS](#)，如 <https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos> 中所述。

如果您使用的是 .NET Core 1.0，则 .NET Core [在 macOS 上使用 OpenSSL](#)，这是一个必须单独安装的依赖项。OpenSSL 在 1.0.1 版本中增加了对 TLS 1.2 的支持，在 1.1.1 版本中增加了对 TLS 1.3 的支持。

Linux

Linux 上的 .NET Core 需要 OpenSSL，它与许多 Linux 发行版捆绑在一起。但也可以单独安装它。OpenSSL 在 1.0.1 版本中增加了对 TLS 1.2 的支持，在 1.1.1 版本中增加了对 TLS 1.3 的支持。如果您使用的是 .NET Core 的最新版本（2.1 或更高版本），并且已安装了程序包管理器，则可能已经为您安装了更高版本的 OpenSSL。

当然，您可以在终端中运行 `openssl version` 并验证版本是否低于 1.0.1。

NET Framework。

如果您正在运行 .NET Framework 的最新版本（4.7 或更高版本）和 Windows 的最新版本（对于客户端，至少为 Windows 8；对于服务器，则为 Windows Server 2012 或更高版本），则默认情况下启用并使用 TLS 1.2。

如果您使用的是不使用操作系统设置的 .NET Framework 运行时（.NET Framework 3.5 到 4.5.2），AWS SDK for .NET 将尝试向支持的协议[添加对 TLS 1.1 和 TLS 1.2 的支持](#)。如果您使用的是 .NET Framework 3.5，则只有在安装了适当的热补丁时才会成功，如下所示：

- Windows 10 版本 1511 和 Windows Server 2016 – [KB3156421](#)
- Windows 8.1 和 Windows Server 2012 R2 – [KB3154520](#)
- Windows Server 2012 – [KB3154519](#)
- Windows 7 SP1 和 Server 2008 R2 SP1 – [KB3154518](#)

Warning

从 2024 年 8 月 15 日起，他们 AWS SDK for .NET 将终止对 .NET Framework 3.5 的支持，并将 .NET Framework 的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化 AWS SDK for .NET](#)。

如果你的应用程序在 Windows 7 SP1 或 Windows Server 2008 R2 SP1 的较新的 .NET 框架上运行，则需要确保注册表中启用 TLS 1.2 支持，如 <https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings> #tls -12 中所述。较新版本的 Windows [在默认情况下已启用它](#)。

有关在 .NET 框架中使用 TLS 的详细最佳实践，请参阅微软的文章 <https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls>。

AWS Tools for PowerShell

[AWS Tools for PowerShell](#) 使用 AWS SDK for .NET 来实施所有对 AWS 服务的调用。环境的行为取决于 PowerShell 你运行的 Windows 版本，如下所示。

Windows PowerShell 2.0 到 5.x

Windows PowerShell 2.0 到 5.x 在 .NET 框架上运行。您可以使用以下命令验证正在使用哪个 .NET 运行时 (2.0 或 4.0)。PowerShell

```
$PSVersionTable.CLRVersion
```

- 使用 .NET Runtime 2.0 时，请按照之前提供的有关 AWS SDK for .NET 和 .NET Framework 3.5 的说明进行操作。

Warning

从 2024 年 8 月 15 日起，他们 AWS SDK for .NET 将终止对 .NET Framework 3.5 的支持，并将 .NET Framework 的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化 AWS SDK for .NET](#)。

- 使用 .NET Runtime 4.0 时，请按照之前提供的有关 AWS SDK for .NET 和 .NET Framework 4+ 的说明进行操作。

Windows PowerShell 6.0

Windows PowerShell 6.0 及更高版本可在 .NET Core 上运行。您可以通过运行以下命令验证正在使用哪个版本的 .NET Core。

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.FrameworkName], $true).FrameworkName
```

按照之前提供的有关AWS SDK for .NET和相关 .NET Core 版本的说明进行操作。

Xamarin

对于 Xamarin，请参阅 <https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security> 上的说明。总而言之：

对于 Android

- 需要 Android 5.0 或更高版本。
- 项目属性，安卓选项：HttpClient 实现必须设置为安卓，SSL/TLS 实现必须设置为原生 TLS 1.2+。

对于 iOS

- 需要 iOS 7 或更高版本。
- 项目属性，iOS Build: HttpClient 实现必须设置为 NS URLSession。

对于 macOS

- 需要 macOS 10.9 或更高版本。
- “项目选项”、“构建”、“Mac Build：HttpClient 实现”必须设置为 NS URLSession。

Unity

您必须使用 Unity 2018.2 或更高版本，并使用 .NET 4.x 等效脚本运行时。您可以在“项目设置”、“配置”、“播放器”中进行设置，如 <https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html> 中所述。.NET 4.x 等效脚本运行时对运行 Mono 或 IL2CPP 的所有 Unity 平台启用 TLS 1.2 支持。欲了解更多信息，请参阅 <https://blog.unity.com/technology/scripting-runtime-improvements-in-unity-2018-2>。

浏览器 (适用于 Blazor WebAssembly)

WebAssembly 在浏览器而不是服务器上运行，并使用浏览器处理 HTTP 流量。因此，TLS 支持由浏览器支持确定。

在 ASP.NET Core 3.1 的预览版中，只有支持的浏览器才支持 Blaz WebAssembly or WebAssembly，如 <https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms> 中所述。所有主流浏览

器在支持之前都支持 TLS 1.2 WebAssembly。如果您的浏览器是这种情况，那么如果您的应用程序运行，它可以通过 TLS 1.2 进行通信。

有关更多信息和验证，请参阅浏览器的文档。

Amazon S3 加密客户端迁移

此主题介绍了如何将应用程序从 Amazon Simple Storage Service (Amazon S3) 加密客户端的版本 1 (V1) 迁移到版本 2 (V2)，并确保应用程序在整个迁移过程中的可用性。

使用 V2 客户端加密的对象无法使用 V1 客户端解密。为了便于迁移到新客户端，而不必同时重新加密所有对象，我们提供了“V1 过渡”客户端。此客户端可以解密 V1 和 V2 加密的对象，但只能加密与 V1 兼容的格式的对象。V2 客户端可以解密 V1 和 V2 加密的对象（当为 V1 对象启用时），但只能加密与 V2 兼容的格式的对象。

迁移概述

这种迁移分三个阶段进行。此处将介绍这些阶段，稍后将详细介绍。在下一阶段开始之前，必须完成所有使用共享对象的客户端的每个阶段。

1. 将现有客户端更新为 V1 过渡客户端以读取新格式。首先，更新您的应用程序，使其依赖于 V1 过渡客户端，而不是 V1 客户端。V1 过渡客户端使您的现有代码能够解密新 V2 客户端编写的对象和以 V1 兼容格式编写的对象。

Note

V1 过渡客户端仅用于迁移目的。移至 V1 过渡客户端后，继续升级到 V2 客户端。

2. 将 V1 过渡客户端迁移到 V2 客户端以编写新格式。接下来，将应用程序中的所有 V1 过渡客户端替换为 V2 客户端，并将安全配置文件设置为 V2AndLegacy。在 V2 客户端上设置此安全配置文件可使这些客户端解密以 V1 兼容格式加密的对象。
3. 更新 V2 客户端，使其不再读取 V1 格式。最后，在所有客户端都迁移到 V2 并且所有对象都已以 V2 兼容格式加密或重新加密之后，请将 V2 安全配置文件设置为 V2 而不是 V2AndLegacy。这可以防止解密 V1 兼容格式的对象。

将现有客户端更新为 V1 过渡客户端以读取新格式

V2 加密客户端使用旧版本客户端不支持的加密算法。迁移的第一步是更新您的 V1 解密客户端，以便它们可以读取新格式。

V1 过渡客户端使您的应用程序能够解密 V1 和 V2 加密的对象。此客户端是 [Amazon.Extensions.S3.Encryption 软件包的一部分](#)。NuGet 在每个应用程序上执行以下步骤以使用 V1 过渡客户端。

1. 接受 [Amazon.Extensions.S3.Encryption](#) 程序包的依赖项。如果您的项目直接依赖于 AWSSDK.S3 或 AWSSDK.KeyManagementService 软件包，您必须更新这些依赖项或将其删除，以便将它们的更新版本与这个新软件包一起引入。
2. 将相应的 using 语句从 Amazon.S3.Encryption 更改为 Amazon.Extensions.S3.Encryption，如下所示：

```
// using Amazon.S3.Encryption;  
using Amazon.Extensions.S3.Encryption;
```

3. 重新构建并重新部署您的应用程序。

V1 过渡客户端与 V1 客户端的 API 完全兼容，因此无需更改其它代码。

将 V1 过渡客户端迁移到 V2 客户端以写入新格式

V2 客户端是 [Amazon.Extensions.S3.Encryption](#) NuGet 软件包的一部分。它使您的应用程序能够解密 V1 和 V2 加密的对象（如果配置为这样做），但只能加密 V2 兼容格式的对象。

更新现有客户端以读取新的加密格式后，您可以继续将应用程序安全更新到 V2 加密和解密客户端。在每个应用程序上执行以下步骤以使用 V2 客户端：

1. 将 EncryptionMaterials 更改为 EncryptionMaterialsV2。
 - a. 使用 KMS 时：
 - i. 提供 KMS 密钥 ID。
 - ii. 声明您正在使用的加密方法；即 KmsType.KmsContext。
 - iii. 向 KMS 提供与该数据密钥关联的加密上下文。您可以发送空字典（Amazon 加密上下文仍将合并到其中），但鼓励提供更多上下文。
 - b. 使用用户提供的密钥封装方法（对称或非对称加密）时：
 - i. 提供包含加密材料的 AES 或 RSA 实例。
 - ii. 声明要使用哪种加密算法；也就是说，SymmetricAlgorithmType.AesGcm 还是 AsymmetricAlgorithmType.RsaOaepSha1。

2. 将 `AmazonS3CryptoConfiguration` 更改为 `AmazonS3CryptoConfigurationV2` , `SecurityProfile` 属性设置为 `SecurityProfile.V2AndLegacy`。
3. 将 `AmazonS3EncryptionClient` 更改为 `AmazonS3EncryptionClientV2`。此客户端采用前面步骤中新转换的 `AmazonS3CryptoConfigurationV2` 和 `EncryptionMaterialsV2` 对象。

示例：KMS 到 KMS+Context

迁移前

```
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);
```

迁移后

```
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);
```

示例：对称算法 (AES-CBC 到 AES-GCM 密钥封装)

StorageMode 可以是 ObjectMetadata 或 InstructionFile。

迁移前

```
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);
```

迁移后

```
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

使用 AES-GCM 解密时，在开始使用解密的数据之前，请通读整个对象。这是为了验证对象自加密以来是否未对其进行过修改。

示例：非对称算法 (RSA 到 RSA-OAEP-SHA1 密钥包装)

StorageMode 可以是 ObjectMetadata 或 InstructionFile。

迁移前

```
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);
```

迁移后

```
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);
```

将 V2 客户端更新为不再读取 V1 格式

最终，所有对象都将使用 V2 客户端进行加密或重新加密。转换完成后，您可以通过将 `SecurityProfile` 属性设置为 `SecurityProfile.V2`，在 V2 客户端中禁用 V1 兼容性，如以下代码片段所示。

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

AWS SDK for .NET的特殊注意事项

本部分包含普通配置或程序不恰当或不够充分的特殊情况下的注意事项。

主题

- [获取用于AWS SDK for .NET的程序集](#)
- [访问应用程序中的凭证和配置文件](#)
- [Unity 支持的特殊注意事项](#)
- [Xamarin 支持的特殊注意事项](#)

获取用于AWS SDK for .NET的程序集

本主题介绍如何获取 AWSSDK 程序集并将其存储在本地（或本地），以便在项目中使用。这不是处理软件开发工具包参考的推荐方法，但在某些环境中却是必需的。

Note

处理 SDK 引用的推荐方法是仅下载并安装每个项目所需的 NuGet 软件包。[通过 NuGet 安装 AWSSDK 程序包](#)中介绍了这种方法。

如果您不能或不允许您按项目下载和安装 NuGet 软件包，则可以使用以下选项。

下载并解压缩 ZIP 文件

(请记住，这不是处理对 AWS SDK for .NET 的引用的[推荐方法](#)。)

1. 下载以下 ZIP 文件之一：

- [aws-sdk-net8.0.zip](#)-支持.NET 8 及更高版本的程序集。
- [aws-sdk-netcoreapp3.1.zip](#)-支持.NET Core 3.1 及更高版本的程序集。
- [aws-sdk-netstandard2.0.zip](#)-支持.NET 标准 2.0 和 2.1 的程序集。
- [aws-sdk-net45.zip](#)-支持.NET 框架 4.5 及更高版本的程序集。
- [aws-sdk-net35.zip](#)-支持.NET 框架 3.5 的程序集。

⚠ Warning

从 2024 年 8 月 15 日起，他们 AWS SDK for .NET 将终止对 .NET Framework 3.5 的支持，并将 .NET 框架的最低版本更改为 4.6.2。有关更多信息，请参阅博客文章 [.NET Framework 3.5 和 4.5 目标即将发生的重要变化 AWS SDK for .NET](#)。

2. 将程序集解压缩到文件系统上的某个“下载”文件夹；什么位置都可以。记下此文件夹。
3. 在设置项目时，可以从该文件夹中获取所需的程序集，如 [在没有 NuGet 的情况下安装 AWSSDK 程序集](#) 中所述。

访问应用程序中的凭证和配置文件

使用凭证的首选方法是允许 AWS SDK for .NET 为您查找和检索凭证，如 [凭证和配置文件解析](#) 中所述。

但是，您还可以将应用程序配置为主动检索配置文件和凭证，然后在创建 AWS 服务客户端时明确使用这些凭证。

要主动检索配置文件和凭证，请使用 [Amazon.Runtime.CredentialManagement](#) 命名空间中的类。

- 要在使用 AWS 凭证文件格式（默认位置的共享 AWS 凭证文件或自定义凭证文件）的文件中查找配置文件，请使用 [SharedCredentialsFile](#) 类。为简洁起见，这种格式的文件有时在此文本中简称为凭证文件。
- 要在 SDK Store 中查找配置文件，请使用 [NetSDKCredentialsFile](#) 类。
- 要同时在凭证文件和 SDK Store 中进行搜索，根据类属性的配置，请使用 [CredentialProfileStoreChain](#) 类。

您可以使用该类来查找配置文件。您也可以使用该类直接请求 AWS 凭证，而不是使用 [AWSCredentialsFactory](#) 类（如下所述）。

- 要从配置文件中检索或创建各种类型的凭证，请使用 [AWSCredentialsFactory](#) 类。

以下部分提供了这些类的示例。

CredentialProfileStoreChain 类的示例

可使用 [TryGetAWSCredentials](#) 或 [TryGetProfile](#) 方法获取 [CredentialProfileStoreChain](#) 类的凭证或配置文件。类的 `ProfilesLocation` 属性将决定方法的行为，如下所示：

- 如果 ProfilesLocation 为 null 或空，则在平台支持的情况下搜索 SDK Store，然后在默认位置搜索共享 AWS 凭证文件。
- 如果 ProfilesLocation 属性包含值，请搜索该属性中指定的凭证文件。

从 SDK Store 或共享 AWS 凭证文件获取凭证

本示例向您展示了如何使用 CredentialProfileStoreChain 类获取凭证，然后使用这些凭证创建 [AmazonS3Client](#) 对象。这些凭证可以来自 SDK Store，也可以来自默认位置的共享 AWS 凭证

本示例还使用了 [Amazon.Runtime.AWSCredentials](#) 类。

```
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}");
    }
}
```

从 SDK Store 或共享 AWS 凭证文件获取配置文件

本示例说明如何使用 CredentialProfileStoreChain 类获取配置文件。这些凭证可以来自 SDK Store，也可以来自默认位置的共享 AWS 凭证

本示例还使用了 [CredentialProfile](#) 类。

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

从自定义凭证文件获取凭证

本示例说明如何使用 CredentialProfileStoreChain 类获取凭证。凭证来自使用 AWS 凭证文件格式但位于备用位置的文件。

本示例还使用了 [Amazon.Runtime.AWSCredentials](#) 类。

```
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
}
```

SharedCredentialsFile 和 AWSCredentialsFactory 类的示例

使用 SharedCredentialsFile 类创建 AmazonS3Client

此示例向您展示了如何在共享 AWS 凭证文件中查找配置文件，根据该配置文件创建 AWS 凭证，然后使用这些凭证创建 [AmazonS3Client](#) 对象。该示例使用 [SharedCredentialsFile](#) 类。

此示例还使用了 [CredentialProfile](#) 类和 [Amazon.Runtime.AWSCredentials](#) 类。

```
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

[NetSDKCredentialsFile](#) 类的使用方式完全相同，唯一的不同是您要实例化一个新的 [NetSDKCredentialsFile](#) 对象，而不是 [SharedCredentialsFile](#) 对象。

Unity 支持的特殊注意事项

在 Unity 应用程序中使用 AWS SDK for .NET 和 [.NET Standard 2.0](#) 时，您的应用程序必须直接引用 AWS SDK for .NET 程序集（DLL 文件），而不是使用 NuGet。鉴于此要求，以下是您需要执行的重要操作。

- 您需要获取 AWS SDK for .NET 程序集并将其应用于您的项目。有关如何操作的信息，请参阅[获取 AWSSDK 程序集](#)主题中的[下载并解压缩 ZIP 文件](#)。
- 您需要在 Unity 项目中加入以下 DLL，以及 AWSSDK.Core 和您正在使用的其它 AWS 服务的 DLL。从 AWS SDK for .NET 版本 3.5.109 开始，.NET Standard ZIP file 文件包含这些其它的 DLL。
 - [Microsoft.Bcl.AsyncInterfaces.dll](#)
 - [System.Runtime.CompilerServices.Unsafe.dll](#)
 - [System.Threading.Tasks.Extensions.dll](#)
- 如果您使用 [IL2CPP](#) 构建 Unity 项目，则必须向 Asset 文件夹中添加 link.xml 文件以防止代码剥离。link.xml 文件必须列出您正在使用的所有 AWSSDK 程序集，并且每个程序集都必须包含 preserve="all" 属性。下面的代码片段显示了此示例文件。

```
<linker>
  <assembly fullname="AWSSDK.Core" preserve="all"/>
  <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
  <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

要阅读与此要求相关的有趣背景信息，请参阅位于以下位置的文章：<https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/>。

除了这些特殊注意事项外，有关将 Unity 应用程序迁移到 AWS SDK for .NET 版本 3.5 的信息，请参阅[3.5 版的更改内容](#)。

Xamarin 支持的特殊注意事项

(新的和现有的) Xamarin 项目必须指向 .NET Standard 2.0。请参阅 [Xamarin.Forms 中的 .NET Standard 2.0 支持](#)和 [.NET 实现支持](#)。

另请参阅有关[便携式类库和 Xamarin](#) 的信息。

AWS SDK for .NET的 API 参考。

AWS SDK for .NET提供了可用于访问 AWS 服务的 API。要查看 API 中有哪些类和方法可用，请参阅[AWS SDK for .NET API 参考](#)。

除了上面给出的一般参考之外，[带有指导的代码示例](#)部分下的每个示例都包含对该示例中使用的特定方法和类的引用。

文档历史记录

下表描述了自上次发布《AWS SDK for .NET 用户指南》以来的重要更改。如需获取对此文档的更新的通知，您可以订阅 [RSS 源](#)。

变更	说明	日期
新增功能	包含有关 .NET Framework 支持即将发生的变更的信息。	2024年2月18日
获取 AWSSDK 程序集	包含有关支持 .NET 8 及更高版本的程序集的信息。	2024 年 1 月 8 日
AWS Message Processing Framework for .NET	包含有关 Message Processing Framework 预览版的信息。	2023 年 12 月 10 日
AWS OpsWorks	添加了有关 AWS OpsWorks 生命周期终止的注释。	2023 年 12 月 8 日
使用 Amazon DynamoDB NoSQL 数据库	更新了有关文档和对象持久性编程模型的信息。现在可以防止由于冷启动和线程池行为而导致的某些延迟或死锁情况。	2023 年 11 月 15 日
加入了更多 IAM 最佳实践更新	更新了指南，使其符合 IAM 最佳实践。有关更多信息，请参阅 IAM 安全最佳实践 。	2023 年 10 月 5 日
获取 AWSSDK 程序集	删除有关使用 AWS Tools for Windows 安装程序 (即 MSI , 已弃用) 安装 AWS SDK for .NET 的信息。	2023 年 9 月 25 日
IAM 最佳实践更新	更新了指南，使其符合 IAM 最佳实践。有关更多信息，请参阅 IAM 安全最佳实践 。	2023 年 7 月 18 日

Lambda 注释	AWS Lambda 注释框架已正式发布。	2023 年 7 月 17 日
新增功能	添加了有关 Distributed Cache Provider for DynamoDB 预览版发布的信息。	2023 年 7 月 15 日
目录	更新了目录，使代码示例更易于发现。	2023 年 6 月 8 日
区域解析	添加了有关 SDK 如何解决缺失的区域规范的信息。	2023 年 3 月 14 日
对 MSI 的支持	添加了有关终止对 AWS Tools for Windows 安装程序的支持的备注。	2023 年 3 月 6 日
Lambda 注释 (预览版)	AWS Lambda 注释框架的预览版。	2022 年 9 月 22 日
将应用程序部署到 AWS	已将主要内容移至 P GitHub ages 网站： https://aws.github.io/aws-dotnet-deploy/	2022 年 6 月 28 日
停用 EC2-Classic	添加了有关停用 EC2-Classic 的说明。	2022 年 4 月 13 日
通过 AWS SDK for .NET 进行单点登录	添加有关使用 AWS SDK for .NET 时进行单点登录 (SSO) 的信息。	2022 年 3 月 17 日
强制实施最低 TLS 版本	添加了有关 TLS 1.3 的信息。	2022 年 3 月 16 日
使用 AWS 服务	包括上可用的代码示例列表 GitHub。	2022 年 2 月 28 日
启用 SDK 指标	删除有关启用 SDK 指标的信息，这些指标已弃用。	2022 年 1 月 20 日

将应用程序部署到 AWS	添加了对 Visual Studio 的 AWS Toolkit for Visual Studio 的引用，该工具包提供的部署功能与 AWS 部署工具类似。	2021 年 10 月 26 日
AWS SDK for .NET 版本 3 指南合并	两个 AWS SDK for .NET 版本 3 开发人员指南，“V3”和“最新”已在“v3”URL 下合并为一个指南。	2021 年 8 月 18 日
从 .NET Standard 1.3 迁移	对 AWS SDK for .NET 上的 .NET Standard 1.3 的支持将结束。	2021 年 3 月 25 日
将应用程序部署到 AWS (预览)	添加了有关 AWS 部署工具的预览信息，您可以使用该工具从 .NET CLI 部署应用程序。	2021 年 3 月 15 日
AWS SDK for .NET 3.5 版	AWS SDK for .NET 3.5 版已经发布。	2020 年 8 月 25 日
分页工具	为许多服务客户端添加了分页工具，这会让 API 结果的分页更加方便。	2020 年 8 月 24 日
重试和超时	添加了有关重试模式的信息。	2020 年 8 月 20 日
S3 加密客户端迁移	添加了有关如何将 Amazon S3 加密客户端从 V1 迁移到 V2 的信息。	2020 年 8 月 7 日
使用 KMS 密钥进行 S3 加密	更新示例，以使用 S3 加密客户端的版本 2。	2020 年 8 月 6 日
从 .NET Standard 1.3 迁移	添加了有关在 2020 年底终止对 .NET Standard 1.3 的支持的信息。	2020 年 5 月 18 日

[快速入门](#)

添加了包含基本设置和教程的快速入门部分，向读者介绍 AWS SDK for .NET。

2020 年 3 月 27 日

[强制执行 TLS 1.2](#)

添加了有关如何在开发工具包中强制执行 TLS 1.2 的信息。

2020 年 3 月 10 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。