aws

开发人员指南

适用于 .NET 的 AWS SDK (V4)

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

适用于 .NET 的 AWS SDK (V4): 开发人员指南

Copyright © 2026 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

..... xi
什么是 适用于 .NET 的 AWS SDK 1
关于本版本 1
SDK 主要版本的维护和支持 2
常见使用案例 2
此部分中的其他主题 2
相关 AWS 工具 3
适用于 Windows 的工具 PowerShell 和适用于 PowerShell 核心的工具 3
Toolkit for VS Code 3
Toolkit for Visual Studio 3
Azure 版工具包 DevOps 3
支持的平台 4
.NET 内核 4
.NET Standard 2.0 4
.NET Framework 4.5 4
.NET Framework 3.5 4
便携式类库和 Xamarin 4
Unity 支持 5
SDKs 和工具参考 5
修订历史记录 5
新增内容 5
其他资源 7
入门 9
安装和配置您的工具链 9
跨平台开发 9
在 Windows 上使用 Visual Studio 和 .NET Core 开发 10
后续步骤 10
使用进行身份验证 AWS 10
使用控制台凭据登录 11
启用和配置 IAM Identity Center 12
将开发工具包配置为使用 IAM Identity Center。 12
启动 AWS 访问门户会话 13
附加信息 13
创建简单的应用程序 14

简单跨平台应用程序 14
基于 Windows 的简单应用程序 19
后续步骤 25
配置 26
启动新项目 26
使用安装 AWSSDK 软件包 NuGet 27
NuGet 从命令提示符或终端使用 28
使用 Vis NuGet ual Studio 解决方案资源管理 29
NuGet 从 Package Manager 控制台中使用 29
安装 AWSSDK 程序集时不使用 NuGet 30
AWS 区域 31
创建具有特定区域的服务客户端 31
为所有服务客户端指定区域 33
区域解析 34
有关中国（北京）区域的特殊信息 34
有关新 AWS 服务的特殊信息 34
凭证和配置文件解析 34
配置文件解析 35
使用联合用户账户凭证 36
指定角色或临时凭证 36
使用代理凭证 37
重试和超时 37
重试 38
超时 39
可观测性 41
其他资源 41
配置 TelemetryProvider 42
Metrics 43
遥测提供商 45
用户和角色 46
用户和权限集 46
服务角色 47
高级配置 47
AWSSDK.Extensions.NETCore.Setup 和 iconfiguration 48
配置其他应用程序参数 52
的配置文件参考 适用于 .NET 的 AWS SDK 60

使用旧版凭证 71
有关凭证的重要警告和指南 72
使用共享 AWS 凭据文件 73
使用 SDK Store（仅适用于 Windows） 76
使用 SDK 80
异步编程 80
性能最佳实践 82
重复使用服务客户端 82
处置响应和直播 83
async/await 正确使用 84
配置垃圾收集 85
管理 HTTP 连接 86
配置超时和重试 87
优化大额转账 88
诊断问题 88
分页 89
我在哪里可以找到分页工具？ 89
分页工具能提供什么好处？ 89
同步分页与异步分页 90
示例 90
分页工具的其它注意事项 93
对 HTTP 2 的支持 94
额外注意事项 96
其他工具 97
AWS 部署工具 97
AWS .NET 的消息处理框架 97
与.NET Aspire 的集成 97
高级身份验证 98
单点登录 98
先决条件 99
设置 SSO 配置文件 99
生成和使用 SSO 令牌 100
其他资源 105
教程 105
教程：仅限 .NET 应用程序 105
教程： AWS CLI 和.NET 应用程序 114

部署到 AWS 122
从 .NET CLI 部署 122
通过 IDE 工具包部署 122
使用案例 123
ASP.NET Core 应用程序 123
.NET Console 应用程序 124
Blazor 应用程序 WebAssembly 124
AWS Lambda 项目 125
先决条件 125
可用的 Lambda 命令 126
部署步骤 126
呼叫 AWS 服务 128
引导式代码示例 128
CloudFormation 129
Amazon Cognito 133
DynamoDB 140
Amazon EC2 168
IAM 227
Amazon S3 245
Amazon SNS 255
Amazon SQS 259
AWS Lambda 292
APIs 292
先决条件 292
其他信息 292
主题 292
Lambda 注释 292
高级库和框架 294
Message Processing Framework 294
集成.NE AWS T Aspire 313
其它服务和配置 314
代码示例 315
Aurora 316
开始使用 316
基本功能 318
操作 342

Auto Scaling 358
开始使用 316
基本功能 318
操作 342
Amazon Bedrock 393
开始使用 316
操作 342
Amazon Bedrock 运行时系统 397
开始使用 316
Anthropic Claude 400
Cohere Command 403
Meta Llama 407
Mistral AI 411
Stable Image Core 414
CloudFormation 416
开始使用 316
CloudWatch 420
开始使用 316
基本功能 318
操作 342
CloudWatch 日志 478
操作 342
场景 481
Amazon Cognito 身份提供者 503
操作 342
AWS Control Tower 505
开始使用 316
基本功能 318
操作 342
DynamoDB 544
开始使用 316
基本功能 318
操作 342
Amazon EC2 584
开始使用 316
Amazon ECS 586

开始使用 316
AWS IoT 588
开始使用 316
基本功能 318
操作 342
AWS IoT data 647
操作 342
Amazon Redshift 650
开始使用 316
基本功能 318
操作 342
Amazon S3 680
开始使用 316
基本功能 318
操作 342
场景 481
迁移您的项目 717
迁移到版本 4 717
NET Framework。 717
值类型 717
集合 718
AWS Security Token Service (STS) 718
与以下内容相关的更改 ClientConfig 719
AWSSDK.Extensions NETCore.Setup 软件包 NuGet 719
CookieSigner 和 UrlSigner 720
DateTime 对阵世界标准时间 DateTime 720
DateTime 正在解析 721
ConvertFromUnixEpochSeconds 和 ConvertFromUnixEpochMilliseconds 721
日志记录 721
对 HTTP 2 的支持 722
单点登录 722
特定于 DynamoDB 的更改 722
特定于 EC2 的更改 725
特定于 S3 的更改 726
已移除的编程元素 729
从 .NET Standard 1.3 迁移 732

安全性 733
数据保护 733
身份和访问管理 734
受众 734
使用身份进行身份验证 735
使用策略管理访问 736
操作方法 AWS 服务 与 IAM 合作 737
问题排查 AWS 身份和访问权限 738
合规性验证 739
恢复能力 740
基础设施安全性 740
强制实施最低 TLS 版本 740
.NET 内核 741
NET Framework。 742
AWS Tools for PowerShell 742
Xamarin 743
Unity 744
浏览器（适用于 Blazor WebAssembly） 744
S3 加密客户端迁移（V1 到 V2） 744
迁移概述 745
将现有客户端更新为 V1-transitional 客户机以阅读新格式 745
将 V1-transitional 客户端迁移到 V2 客户端以编写新格式 746
将 V2 客户端更新为不再读取 V1 格式 749
S3 加密客户端迁移（V2 到 V4） 749
理解 V4 概念 750
更新现有客户端以读取 V4 格式 751
迁移到 V4 客户端 752
其他配置示例 755
特殊注意事项 758
获取 AWSSDK 程序集 758
下载并解压缩 ZIP 文件 758
访问应用程序中的凭证和配置文件 759
课堂示例 CredentialProfileStoreChain 759
类 SharedCredentialsFile 和 AWSCredentials工厂的示例 761
Unity 支持 761
Xamarin 支持 762

API 参考 763
关于 API 参考版本 763
文档历史记录 765

的第 4 版 (V4) 适用于 .NET 的 AWS SDK 已经发布！

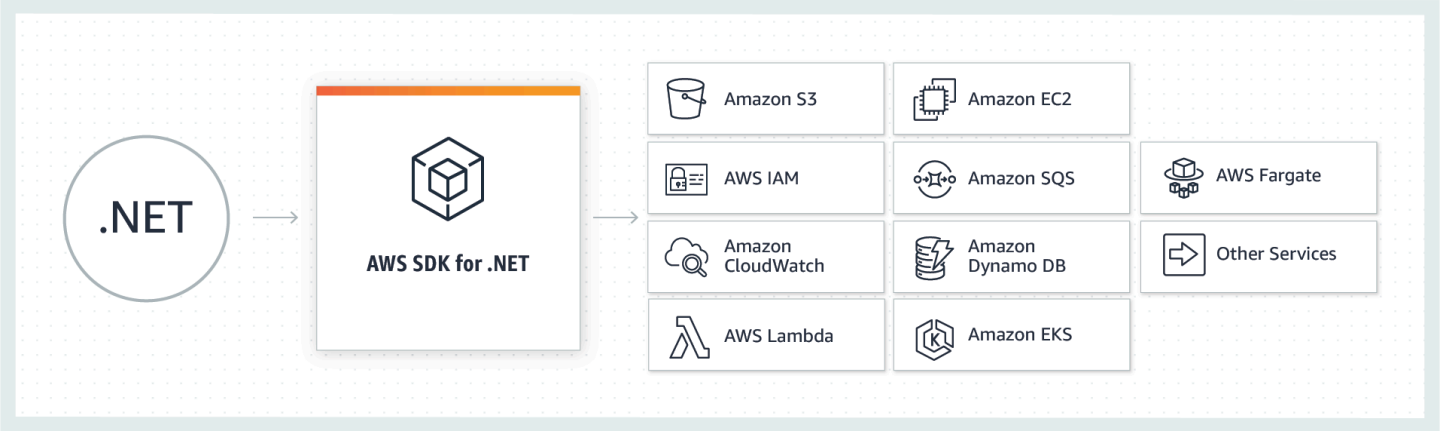
有关重大更改和迁移应用程序的信息，请参阅迁移主题。

Click here for details

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。

什么是 适用于 .NET 的 AWS SDK

适用于 .NET 的 AWS SDK 这使得构建.NET 应用程序变得更加容易，这些应用程序可以利用经济高效、可扩展和可靠的 AWS 服务，例如亚马逊简单存储服务 (Amazon S3) 和亚马逊弹性计算云 (Amazon EC2)。该开发工具包为 .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！ 我已经准备好进行设置并快速浏览了。）

关于本版本

当前和过去的文档

本文档适用于 4.0 及更高版本 (V4) 的。 适用于 .NET 的 AWS SDK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。

有关从早期版本的 SDK 迁移的信息，请参阅迁移您的项目。

有关 SDK 的先前版本 V3（将来会被弃用）的信息，请参阅 适用于 .NET 的 AWS SDK (V3) 开发者指南。

要查找早期版本的 SDK 的已弃用内容，请参阅以下内容：

- 适用于 .NET 的 AWS SDK（版本 2，已弃用）开发者指南
- 已弃用的 API 引用 适用于 .NET 的 AWS SDK

SDK 主要版本的维护和支持

有关 SDK 主要版本及其底层依赖项的维护和支持的信息，请参阅《AWS SDKs 和工具参考指南》中的以下内容：

- AWS SDKs 和工具维护政策
- AWS SDKs 和工具版本支持矩阵

常见使用案例

适用于 .NET 的 AWS SDK 可以帮助您实现几个引人注目的用例，包括：

- 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。
- 创建和启动亚马逊 EC2实例，配置和请求亚马逊 EC2 竞价实例。

此部分中的其他主题

- AWS 与之相关的工具 适用于 .NET 的 AWS SDK
- 支持的平台 适用于 .NET 的 AWS SDK
- AWS SDKs 和《工具参考指南》
- 修订历史记录
- 里面有哪些新内容 适用于 .NET 的 AWS SDK
- 其他资源

AWS 与之相关的工具 适用于 .NET 的 AWS SDK

适用于 Windows 的工具 PowerShell 和适用于 PowerShell 核心的工具

AWS Tools for Windows PowerShell 和 AWS Tools for PowerShell Core 是基于公开的功能构建的 PowerShell 模块 适用于 .NET 的 AWS SDK。这些 AWS PowerShell 工具使您能够在 PowerShell 提示符下编写对 AWS 资源的操作脚本。尽管 cmdlet 是使用 SDK 中的服务客户端和方法实现的，但是 cmdlet 为指定参数和处理结果提供了一种惯用的 PowerShell 体验。

要开始使用，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，然后将这些应用程序部署到 AWS CloudFormation 堆栈。
- 使用 Amazon EventBridge 架构。
- 在处理 Amazon ECS 任务定义文件 IntelliSense 时使用。
- 可视化 AWS Cloud Development Kit (AWS CDK) 应用程序。

Toolkit for Visual Studio

AWS Toolkit for Visual Studio 是 Visual Studio IDE 的插件，可让您更轻松地开发、调试和部署使用亚马逊 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 资源管理器在 Visual Studio 中管理亚马逊 EC2 实例、使用亚马逊 DynamoDB 表、向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 队列发布消息等。

要开始使用，请参阅设置 AWS Toolkit for Visual Studio。

Azure 版工具包 DevOps

AWS Toolkit for Microsoft Azure DevOps 添加了任务，可以轻松启用 Azure 和 Azure DevOps Server 中的生成 DevOps 和发布管道来处理 AWS 服务。你可以使用亚马逊 S3、、、Lambda、 AWS Elastic

Beanstalk AWS CodeDeploy AWS CloudFormation、亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 和亚马逊 SNS。你也可以使用 Windows PowerShell 模块和 AWS Command Line Interface (AWS CLI) 来运行命令。

要开始使用 AWS Toolkit for Azure DevOps，请参阅《AWS Toolkit for Microsoft Azure DevOps 用户指南》。

支持的平台 适用于 .NET 的 AWS SDK

为开发人员 适用于 .NET 的 AWS SDK 提供了针对不同平台的不同程序集组。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

适用于 .NET 的 AWS SDK 支持为.NET Core 编写的应用程序（.NET Core 3.1、.NET 5、.NET 6 等）。 AWS 服务客户端仅支持.NET 核心中的异步调用模式。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

的非框架变体 适用于 .NET 的 AWS SDK 符合 .NET 标准 2.0。 适用于 .NET 的 AWS SDK 仅为根据 .NET Standard 编写的应用程序提供异步方法。

.NET Framework 4.5

此版本的针对.NET 适用于 .NET 的 AWS SDK Framework 4.7.2 进行编译，并在.NET 4.0 运行时中运行。 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

的版本 4 适用于 .NET 的 AWS SDK 不支持.NET 框架 3.5。

便携式类库和 Xamarin

适用于 .NET 的 AWS SDK 还包含可移植类库实现。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 SDK 和 X amarin for 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。

AWS SDKs 和《工具参考指南》

AWS SDKs 和工具参考指南包含与许多和工具包相关 AWS SDKs 且重要的信息，以及. AWS CLI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 常用运行时 (CRT) 库
- AWS SDKs 和工具维护政策
- AWS SDKs 和工具版本支持矩阵

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：

- SDK 变更日志
- 里面有哪些新内容 适用于 .NET 的 AWS SDK
- 文档历史记录

里面有哪些新内容 适用于 .NET 的 AWS SDK

有关与之相关的新开发的高级信息 适用于 .NET 的 AWS SDK，请参阅产品页面https://aws.amazon.com/sdk-for-net/和 SDK 变更日志。

以下是 适用于 .NET 的 AWS SDK中的新增内容。

2025 年 10 月 17 日： AWS 部署工具 2.0 版

适用于.NET CLI 的 AWS 部署工具 2.0 版已经发布。有关部署工具的信息，请参阅将应用程序部署到 AWS。有关部署工具 2.0 版本的信息，请参阅博客文章 “.NET AWS 部署工具的新增功能”。

2025 年 8 月 21 日：第 3 版即将推 end-of-support出 适用于 .NET 的 AWS SDK

end-of-support的版本 3 (V3) 适用于 .NET 的 AWS SDK 已经发布。请参阅迁移指南，迁移您的 V3 应用程序并避免中断。有关更多信息，请参阅博客文章宣布推出 end-of-support 适用于 .NET 的 AWS SDK v3。

2025 年 7 月 2 日：分布式缓存提供程序正式上市

亚马逊 Dy AWS namoDB 的.NET 分布式缓存提供程序现已正式上市！该提供程序实现了 ASP.NET Core C IDistributedache 接口，让您只需最少的代码更改即可将 DynamoDB 的完全托管且持久的基础架构集成到您的缓存层中。分布式缓存提供程序可通过 AWS 获得。 AspNetCore。 DistributedCacheProvider开启包裹 NuGet。有关更多信息，请参阅博客文章 Amazon Dy AWS namoDB 的.NET 分布式缓存提供程序现已正式上市。

> **Note**
>
> 分布式缓存提供程序的发布版本仅在 V4 上受支持。 适用于 .NET 的 AWS SDK如果您尝试使用分布式缓存提供程序的预览版，则可能需要更新应用程序中的软件包依赖关系。

2025 年 5 月 5 日：.NET AWS 消息处理框架正式上市

.NET 的AWS 消息处理框架现已正式上线！该框架是一个 AWS原生框架，可简化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

2025 年 4 月 28 日：第 4 版 适用于 .NET 的 AWS SDK

的版本 4 现 适用于 .NET 的 AWS SDK 已正式上市！有关将应用程序迁移到 V4 的信息，请参阅迁移到版本 4。另请参阅 适用于 .NET 的 AWS SDK V4.0 正式发布的博客文章，其中宣布正式上市。

2025 年 2 月 15 日：与.NET Aspire 集成

已经发布了与.NET Aspire 的集成，以改善内部开发循环。有关信息，请参阅在中 AWS 与.NET Aspire 集成 适用于 .NET 的 AWS SDK。

2025 年 2 月 10 日：GA 发布可观测性

可观测性是指可以从系统发出的数据中推断出其当前状态的程度。可观察性已添加到中 适用于 .NET 的 AWS SDK，包括遥测提供程序的实现。有关更多信息，请参阅可观测性本指南和博客文章宣布 AWS .NET OpenTelemetry 库正式上市。

2025 年 1 月 15 日：诚信保护的新违约行为

从的 3.7.412.0 版本开始 适用于 .NET 的 AWS SDK，SDK 通过自动计算上传的校验和来提供默认的CRC32完整性保护。有关更多信息，请参阅上的 GitHub 公告https://github.com/aws/aws-sdk-net/issues/3610。SDK 还提供了可在外部设置的数据完整性保护的全局设置，您可以在《工具参考指南》AWS SDKs 和《工具参考指南》的《数据完整性保护》中阅读这些设置。

2024 年 11 月 15 日：第 4 版预览版发布的博客文章

第 4 适用于 .NET 的 AWS SDK 版的预览版 4 已于 2024 年 11 月 15 日发布。有关此预览版的更多信息，请参阅博客文章 V 4 的 Pre 适用于 .NET 的 AWS SDK view 4。

2024 年 8 月 16 日：预览版博客文章 1 版本第 4 版

第 4 适用于 .NET 的 AWS SDK 版于 2024 年 8 月 16 日作为首次预览版发布。有关此预览版的更多信息，请参阅博客文章 适用于 .NET 的 AWS SDK V4 的 Preview 1。

其他资源

支持的服务

适用于 .NET 的 AWS SDK 支持大多数 AWS 基础架构产品，并且经常添加更多服务。有关 SDK 支持的 AWS 服务的列表，请参阅 S DK 自述文件。

的主页 适用于 .NET 的 AWS SDK

有关更多信息 适用于 .NET 的 AWS SDK，请参阅 SDK 的主页，网址为https://aws.amazon.com/sdk-for-net/。

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。有关更多信息，请参阅 适用于 .NET 的 AWS SDK API 参考。

AWS re: post（以前是论坛 AWS ）

请访问 AWS re: pos t（特别是主题）适用于 .NET 的 AWS SDK，提出问题或提供反馈。 AWS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。要关注的主题 适用于 .NET 的 AWS SDK，请前往 “所有主题” 页面，找到 “.NET on AWS”，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

请访问 GitHub 网站上的 aws/dotnet 和 aws/ aws-sdk-net 存储库，获取可用于帮助构建.NET 应用程序和服务的库、工具和资源的链接。 AWS

下面是一些示例：

- AWS Systems Manager 的.NET 配置扩展
- AWS 扩展.NET 核心设置
- AWS 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
- .NET AWS 云端开发环境-快速入门参考部署
- Hello, Cloud! 博客
- AWS 白皮书：在上开发和部署.NET 应用程序 AWS
- AWS Microservice Extractor for .NET
- Porting Assistant for .NET
- AWS SDKs 和工具参考指南

开始使用 适用于 .NET 的 AWS SDK

了解如何安装、设置和使用 适用于 .NET 的 AWS SDK。您必须安装工具链，并设置应用程序访问 AWS 服务所需的许多基本内容。这些指令包括：

- 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

有关如何配置项目和其他关键部分的信息，请参阅配置 适用于 .NET 的 AWS SDK。

主题

- 安装和配置您的工具链 适用于 .NET 的 AWS SDK
- 使用进行身份验证 适用于 .NET 的 AWS SDK AWS
- 使用创建简单的应用程序 适用于 .NET 的 AWS SDK

安装和配置您的工具链 适用于 .NET 的 AWS SDK

要使用 适用于 .NET 的 AWS SDK，必须安装某些开发工具。

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。

  例子包括微软 Visual Studio Code（VS Code）、JetBrains Rider 和微软 Visual Studio。

- （可选）一个 AWS 工具包（如果有）适用于您选择的编辑器和操作系统。

  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，它是一个插件，它提供了一个用户界面，用于管理 Visual Studio 中的 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

使用进行身份验证 适用于 .NET 的 AWS SDK AWS

使用进行身份验证 适用于 .NET 的 AWS SDK AWS

使用开发 AWS 时，您必须确定您的代码是如何进行身份验证的。 AWS 服务您可以通过不同的方式配置对 AWS 资源的编程访问权限，具体取决于环境和可用的 AWS 访问权限。

要查看 SDK 的各种身份验证方法，请参阅《工具参考指南》和《工具参考指南》中的身份验证AWS SDKs 和访问权限。

本主题假设新用户正在本地开发，雇主未向其提供身份验证方法，并将使用该用户 AWS IAM Identity Center 来获取临时证书。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. 使用控制台凭据登录
2. 启用和配置 IAM Identity Center
3. 将开发工具包配置为使用 IAM Identity Center。

4. 启动 AWS 访问门户会话

使用控制台凭据登录

您可以使用现有的 AWS 管理控制台登录凭据以编程方式访问 AWS 服务。在基于浏览器的身份验证流程之后， AWS 生成可在本地开发工具（如 CL AWS I AWS Tools for PowerShell 和适用于.NET 的 AWS SDK v4）上使用的临时证书。

通过此流程，您可以使用在初始账户设置期间创建的根证书、IAM 用户或身份提供商提供的联合身份以及 AWS CLI 进行身份验证，或者 AWS Tools for PowerShell 自动为您管理临时证书。这种方法无需在本地存储长期凭证，从而增强了安全性。当您从中运行 `Invoke-AWSLogin` cmdlet AWS Tools for PowerShell 或从 CL AWS I 运行`aws login`命令时，您可以从活动控制台会话中进行选择，也可以通过基于浏览器的身份验证流程登录，这将自动生成临时证书。临时凭证令牌将在 15 分钟后过期，但是在您的请求期间，适用于.NET v4 的 AWS SDK 会在需要时自动刷新令牌。

> **Important**
>
> 如果您使用控制台凭据进行身份验证，则您的应用程序必须引用以下 NuGet 软件包，这样 AWS 登录解析才能起作用：
>
> - AWSSDK. 登录
>
> 未能引用这些程序包将导致运行时系统异常。

- AWS Tools for PowerShell 登录凭据《开发者指南》。
- 使用 AWS CLI 使用控制台凭据登录进行 AWS 本地开发。

按照上面给出的 AWS Tools for PowerShell 或 AWS CLI 步骤登录后，`config`文件将如下所示。

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

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

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 SDK 执行此操作，请查看AWS SDKs 和工具参考指南中 IAM Identity Center 身份验证主题中的步骤 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。

有关如何配置软件开发工具包以使用 IAM Identity Center 的信息，请参阅《工具参考指南》中 IAM 身份中心身份验证主题的步骤 2。AWS SDKs 完成此配置后，您的系统应包含以下元素：

- AWS CLI，用于在运行应用程序之前启动 AWS 访问门户会话。
- 共享 AWS `config`文件，其中包含一个`[default]`配置文件，其中包含一组可从 SDK 中引用的配置值。要查找此文件的位置，请参阅AWS SDKs 和工具参考指南中的共享文件的位置。在向发送请求之前， 适用于 .NET 的 AWS SDK 使用配置文件的 SSO 令牌提供者获取凭证。AWS`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。该`sso-session`部分包含启动 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，则您的应用程序必须引用以下 NuGet软件包，这样 SSO 解析才能起作用：

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

未能引用这些程序包将导致运行时系统异常。

启动 AWS 访问门户会话

在运行可访问的应用程序之前 AWS 服务，您需要为软件开发工具包进行有效的 AWS 访问门户会话，才能使用 IAM Identity Center 身份验证来解析证书。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，请在中运行以下命令 AWS CLI。

```
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`aws sso login`正在运行，则无需提供凭据。
> 登录过程可能会提示您允许 AWS CLI 访问您的数据。由于 AWS CLI 是在适用于 Python 的 SDK 之上构建的，因此权限消息可能包含`botocore`名称的变体。

附加信息

- 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 SDK 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 SDKs 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 证书，请参阅 IAM 用户指南中的临时安全证书。
- 要了解其他凭证提供商，请参阅《工具参考指南》AWS SDKs 中的标准化凭证提供商。

使用创建简单的应用程序 适用于 .NET 的 AWS SDK

本部分为不熟悉 适用于 .NET 的 AWS SDK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 服务开发软件的信息以及代码示例，请参见呼叫 AWS 服务。有关其他代码示例，请参阅 适用于 .NET 的 SDK (v4) 代码示例。

主题

- 简单的跨平台应用程序使用 适用于 .NET 的 AWS SDK
- 基于 Windows 的简单应用程序使用 适用于 .NET 的 AWS SDK
- 后续步骤

简单的跨平台应用程序使用 适用于 .NET 的 AWS SDK

本教程使用 适用于 .NET 的 AWS SDK 和.NET Core 进行跨平台开发。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，请参阅安装和配置您的工具链 适用于 .NET 的 AWS SDK。

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。

例子包括微软 Visual Studio Code（VS Code）、JetBrains Rider 和微软 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.SSOOIDC
```

前面的命令通过 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，所以我们现在可以执行这个步骤了。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，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.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

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

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

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

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

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

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

        //
        // Method to parse the command line.
        private static Boolean GetBucketName(string[] args, out String bucketName)
        {
            Boolean retval = false;
            bucketName = String.Empty;
            if (args.Length == 0)
            {
                Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
```

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

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

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

```
            return response.Arn;
        }
    }
}
```

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。使用 “dotnet-quicktour-s3-1-cross-” 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. 运行以下命令，*amzn-s3-demo-bucket*替换为您选择的存储桶的名称。

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

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存储桶，请使用位于的 Amazon S3 控制台将其删除https://console.aws.amazon.com/s3/。
- 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。

基于 Windows 的简单应用程序使用 适用于 .NET 的 AWS SDK

本教程使用了 Windows 适用于 .NET 的 AWS SDK 上的 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，请参阅安装和配置您的工具链 适用于 .NET 的 AWS SDK。

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 4.7.2 或更高版本。

2. 加载新创建的项目后，选择 “工具”、“Pack NuGet age 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.SSOOIDC`

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，所以我们现在可以执行这个步骤了。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. 如果您打算从命令符提示处运行应用程序，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.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

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

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

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

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

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

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

```
        }

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

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

```
            return credentials;
        }
    }

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

3. 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*amzn-s3-demo-bucket*中的内容替换为您选择的存储桶的名称。

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

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存储桶，请使用位于的 Amazon S3 控制台将其删除https://console.aws.amazon.com/s3/。
- 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。这些可能是开发环境中的 AWS 资源或资源，例如文件和文件夹。

既然你已经参观了 适用于 .NET 的 AWS SDK，你可能想开始你的项目。

配置 适用于 .NET 的 AWS SDK

要使用 适用于 .NET 的 AWS SDK，您需要设置环境并配置应用程序访问 AWS 服务所需的许多基本内容。这些指令包括：

- AWS 区域规格
- AWSSDK 包或组件

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

- 启动新项目
- 使用安装 AWSSDK 软件包 NuGet
- 安装 AWSSDK 程序集时不使用 NuGet
- 为设置 AWS 区域 适用于 .NET 的 AWS SDK
- 凭证和配置文件解析
- 重试和超时
- 可观测性
- 有关用户和角色的其它信息
- 适用于 .NET 的 AWS SDK 项目的高级配置
- 使用旧版凭证

启动新项目

您可以使用多种技术来启动新项目以访问 AWS 服务。以下是一些此类技术：

- 如果您不熟悉.NET 开发， AWS 或者至少是新手 适用于 .NET 的 AWS SDK，则可以在中查看完整的示例创建简单的应用程序。其中提供了对开发工具包的简单介绍。

- 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]
```

创建了一个空项目，您可以向其中添加代码和 NuGet 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。

- 如果你在 Windows 上使用 Visual Studio 进行开发 AWS Toolkit for Visual Studio，但没有使用，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。创建了一个空项目，您可以向其中添加代码和 NuGet包。

您可以在中找到一些有关如何使用 AWS 服务的示例引导式代码示例。

> **Important**
>
> 如果您使用 AWS IAM Identity Center 进行身份验证，则您的应用程序必须引用以下 NuGet软件包，这样 SSO 解析才能起作用：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未能引用这些程序包将导致运行时系统异常。

使用安装 AWSSDK 软件包 NuGet

NuGet是.NET 平台的软件包管理系统。使用 NuGet，您可以将这些AWSSDK软件包以及其他几个扩展安装到您的项目中。有关更多信息，请参阅网站上的 aws/dotnet 存储库。 GitHub

NuGet 总是有最新版本的 AWSSDK 软件包以及以前的版本。 NuGet知道软件包之间的依赖关系并自动安装所有必需的软件包。

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

安装在一起 NuGet 的软件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。有关的更多信息 NuGet，请参阅NuGet 文档。

> **Note**
>
> 如果您不能或不允许您按项目下载和安装 NuGet 软件包，则可以获取 AWSSDK 程序集并将其存储在本地（或本地）。
> 如果这适用于您，并且您尚未获得 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，请参阅安装 AWSSDK 程序集时不使用 NuGet。

NuGet 从命令提示符或终端使用

1. 转到上的AWSSDK 软件包 NuGet并确定您的项目中需要哪些包；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
   ```

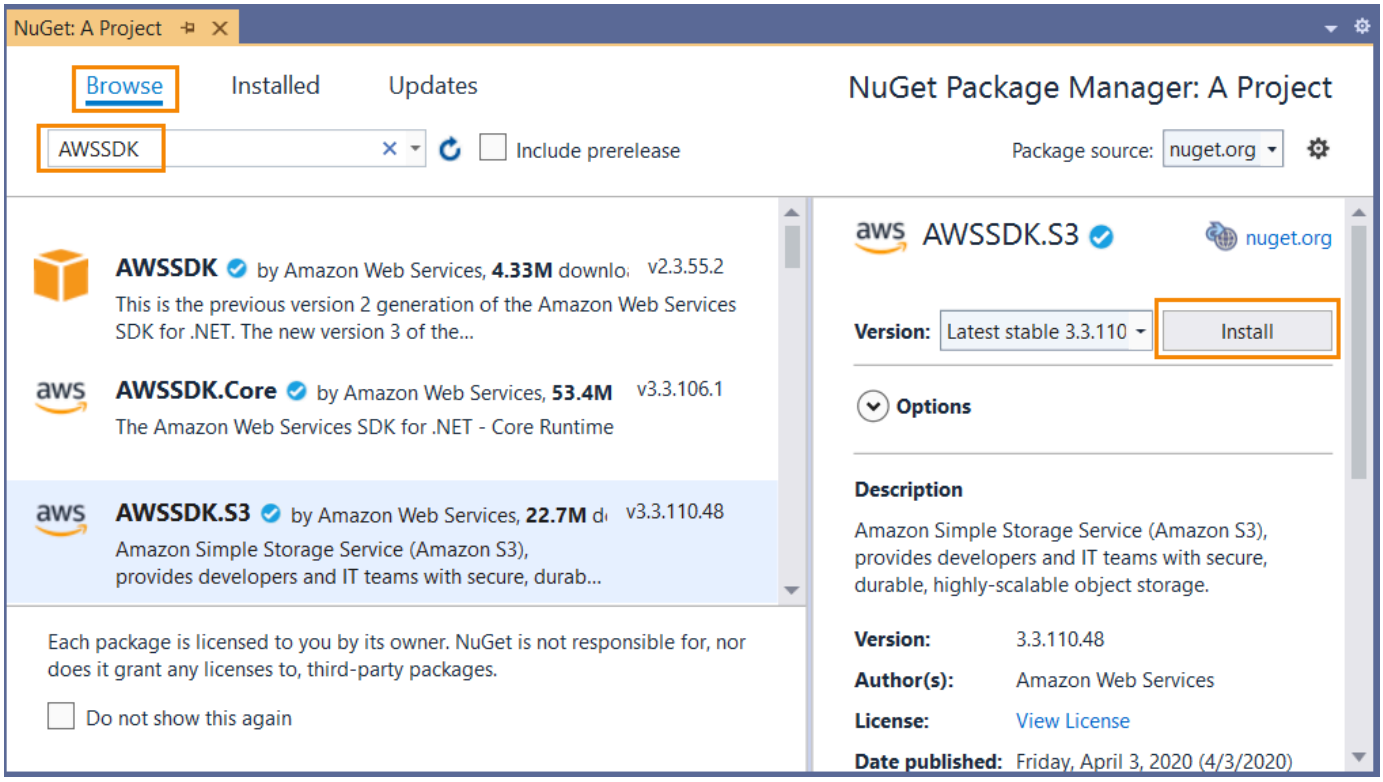
3. 在项目的目录中，运行该.NET CLI 命令。 NuGet 还会安装任何依赖项，例如 AWSSDK.Core。

> **Note**
>
> 如果您只需要最新版本的 NuGet 软件包，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`

使用 Vis NuGet ual Studio 解决方案资源管理

1. 在解决方案资源管理器中，右键单击您的项目，然后从上下文菜单中选择 “管理 NuGet 包”。
2. 在 Package Man NuGet ager 的左侧窗格中，选择 “浏览”。然后，您可以使用搜索框搜索要安装的软件包。 NuGet 还会安装任何依赖项，例如 AWSSDK.Core。

   下图显示了 AWSSDK.S3 软件包的安装。



NuGet 从 Package Manager 控制台中使用

在 Visual Studio 中，选择工具、NuGet 软件包管理器、软件包管理器控制台。

您可以使用**`Install-Package`**命令从 Pac AWSSDK kage Manager 控制台安装所需的软件包。例如，要安装 AWSSDK.S3，请使用以下命令。

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

NuGet 还会安装任何依赖项，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，如以下示例所示。

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

有关 Package Manager 控制台命令的更多信息，请参阅微软NuGet文档中的PowerShell参考资料。

安装 AWSSDK 程序集时不使用 NuGet

本主题介绍如何使用您在本地（或本地）获取并存储的 AWSSDK 程序集，如中所述获取 AWSSDK 程序集。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。使用安装 AWSSDK 软件包 NuGet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. 如果您还没有这样做，请获取 AWSSDK程序集，这会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。你可以在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和将所需的 DLL 文件复制到的子文件夹（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。

为设置 AWS 区域 适用于 .NET 的 AWS SDK

AWS 区域允许您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。

要查看每项 AWS 服务支持的所有区域和终端节点的当前列表，请参阅中的服务终端节点和配额AWS 一般参考。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

此示例向您展示如何在现有区域中实例化 A mazon 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 类创建新区域

此示例向您展示如何使用构造新的区域终端节点RegionEndpoint。 GetBySystemName。

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

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；在本例中为使用 Amazon EC2 Config 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。

按以下顺序查 适用于 .NET 的 AWS SDK 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**
>
> 如果你为整个系统设置这个环境变量（使用export或setx），它会影响所有 SDKs 和工具包，而不仅仅是 适用于 .NET 的 AWS SDK。

AWSConfigs 班级

设为 AWSConfigs. AWSRegion财产。

```
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，则会 适用于 .NET 的 AWS SDK 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，SDK 会抛出异常，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 地区的账户和证书不适用于中国（北京）区域。同样，中国（北京）地区的账户和证书不适用于 AWS 其他地区。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 AWS 服务最初可以在几个地区推出，然后在其他地区提供支持。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。

凭证和配置文件解析

按特定顺序 适用于 .NET 的 AWS SDK 搜索凭证，并使用当前应用程序的第一组可用凭证。

凭证搜索顺序

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

> **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. 从AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY、和AWS_SESSION_TOKEN环境变量创建的@@ 会话 AWSCredentials（如果所有三个变量都有一个值）。
3. 从AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY环境变量创建@@ 的基本AWSCredentials变量（如果两个变量都有值）。
4. AssumeRoleWithWebIdentityCredentials如果两个变量都有值，则由AWS_WEB_IDENTITY_TOKEN_FILE和AWS_ROLE_ARN环境变量创建。
5. 凭据配置文件，其名称由中的值指定AWSConfigs。 AWSProfile姓名。
6. 名称由 AWS_PROFILE 环境变量指定的凭证配置文件。
7. [default] 凭证配置文件。
8. 容器凭证提供商。
9. 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，如中所述使用 IAM 角色授予访问权限。否则，例如在预发行测试中，请将您的凭据存储在文件中，该文件使用您的 Web 应用程序在服务器上可以访问的 AWS 凭据文件格式。

有关环境变量和凭据配置文件的更多信息，请参阅《AWS SDKs 和工具参考指南》中的以下主题：环境变量、环境变量列表以及共享配置和凭据文件。

配置文件解析

由于有两种不同的凭证存储机制，因此了解如何配置 适用于 .NET 的 AWS SDK 以使用它们非常重要。的AWSConfigs。 AWSProfiles位置属性控制如何 适用于 .NET 的 AWS SDK 查找凭据配置文件。

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 ~/.aws/config （Linux 或 macOS）或 %USERPROFILE%\.aws\config （Windows）。

AWSProfilesLocation	配置文件解析行为
AWS 凭证文件格式的文件的路径	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

使用 适用于 .NET 的 AWS SDK （ AWSSDK.Core 版本 3.1.6.0 及更高版本 ）的应用程序可以通过 Active Directory 联合身份验证服务 (AD FS) 使用联合用户帐户，通过安全断言标记语言 (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。这些临时证书的有效期为一小时，将在您的应用程序调 AWS 用服务时使用。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。（对于未加入域的帐户，在重新进行身份验证之前，系统会提示用户输入凭据。）

要在.NET 应用程序中使用此支持，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，其中一种是使用AWSConfigs。 AWSProfile使用与其他凭据配置文件相同的方式命名属性。

大AWS Security Token Service会 (AWSSDK. SecurityToken) 提供 SAML 支持以获取 AWS 证书。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

对于在 Amazon EC2 实例上运行的应用程序，管理证书的最安全方法是使用 IAM 角色，如中所述使用 IAM 角色授予访问权限。

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。除了提供有限的 AWS 资源访问权限外，这些证书还具有在指定时间段后过期的好处。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 AWS 通过代理与通信，则可以使用服务Config类的ProxyCredentials属性为代理指定凭据。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：AmazonCloudDirectoryConfig在 Amazon 中。 CloudDirectory命名空间和AmazonGameLiftConfig在 Amazon 中。 GameLift命名空间。

例如，对于 Amazon S3，您可以使用类似于以下内容的代码，其中SecurelyStoredUserName和SecurelyStoredPassword是在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。

重试和超时

适用于 .NET 的 AWS SDK 允许您为向 AWS 服务发出 HTTP 请求配置重试次数和超时值。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：

- 当网络连接下降或无法访问 AWS 服务时， 适用于 .NET 的 AWS SDK 和您的应用程序应如何响应？ 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？

重试和超时设置会影响应用程序的感知性能。有关相关建议，请参阅性能最佳实践。

重试

概述

适用于 .NET 的 AWS SDK 可以重试因服务器端限制或连接中断而失败的请求。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。服务配置类从 适用于 .NET 的 AWS SDK API 参考的抽象Amazon.Runtime.ClientConfig类继承这些属性：

- `RetryMode`指定Amazon.Runtime.RequestRetryMode枚举中定义的三种重试模式之一。

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

- `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。

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

这些属性的详细描述可以在 适用于 .NET 的 AWS SDK API 参考的抽象Amazon.Runtime.ClientConfig类中找到。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	对应 MaxErrorRetry（亚马逊 DynamoDB）	对应 MaxErrorRetry（所有其他）
Legacy	10	4
标准	10	2
自适应（实验性）	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 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

适用于 .NET 的 AWS SDK 使您能够在服务客户端级别和每个方法调用配置请求超时。有两种配置超时的机制，后续章节将介绍这些机制：

- 如果您使用的是异步调用，则可以使用该方法的 CancellationToken 参数。
- 如果您在 .NET 框架中使用同步调用，则可以使用抽象Amazon.Runtime.ClientConfig类的Timeout和ReadWriteTimeout属性。

使用 CancellationToken 参数配置超时

适用于 .NET 的 AWS SDK 允许您使用CancellationToken参数配置异步调用的请求超时。下面的代码片段展示了一个示例。如果请求未在 10 秒内完成，则代码会引发 System.Threading.Tasks.TaskCanceledException。

```
string bucketName = "amzn-s3-demo-bucket";
string path = "pathToBucket";
using (var amazonS3Client = new AmazonS3Client(new AmazonS3Config()))
{
    // Cancel request after 10 seconds
```

```
    CancellationTokenSource cancellationTokenSource = new
 CancellationTokenSource(TimeSpan.FromMilliseconds(10000));
    CancellationToken cancellationToken = cancellationTokenSource.Token;
    ListObjectsV2Request listRequestV2 = new()
    {
        BucketName = bucketName,
        Prefix = path,
    };

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

使用##和ReadWriteTimeout属性进行超时

> **Note**
>
> Timeout 属性不会影响异步调用。如果您使用的是异步调用，请参阅使用 CancellationToken 参数配置超时。

适用于 .NET 的 AWS SDK 使您能够在服务客户端级别配置请求超时和套接字 read/write 超时值。这些值在抽象Amazon.Runtime.ClientConfig类的Timeout和ReadWriteTimeout属性中指定。这些值作为 AWS 服务客户端HttpWebRequest对象创建的对象的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**
>
> 以可移植类库 (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。 适用于 .NET 的 AWS SDK

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如果调用的方法上传流，则将和ReadWriteTimeout设置为最大值，例如 AmazonS3Client.PutObjectAsync ()、 AmazonS3Client.UploadPartAsync ()、AmazonGlacierClient.UploadArchiveAsync () 等。

- 目标.NET 框架设置 适用于 .NET 的 AWS SDK 的版本`Timeout`以及`ReadWriteTimeout`所有 Amazons3Client 和对象的最大值。 AmazonGlacierClient
- 针对可移植类库 (PCL) 和.NET Core 的版本设置`Timeout`为所有 Amazons3 Client 和对象的最大值。 适用于 .NET 的 AWS SDK AmazonGlacierClient

以下示例说明如何指定标准重试模式、最多 3 次重试、10 秒超时和 10 秒 read/write 超时（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 ### .NET # AWS SDK that target .NET Core.
    ReadWriteTimeout = TimeSpan.FromSeconds(10),
    RetryMode = RequestRetryMode.Standard,
    MaxErrorRetry = 3
  });
```

可观测性

可观测性是指可以根据系统发出的数据推断出其当前状态的程度。发出的数据通常称为遥测。

适用于 .NET 的 AWS SDK 可以提供两种常见的遥测信号、指标和轨迹以及日志记录。您可以连接TelemetryProvider以将遥测数据发送到可观测性后端（例如或 AWS X-RayA ma zon CloudWatch），然后对其进行操作。

默认情况下，SDK 中的遥测信号处于禁用状态。本主题介绍如何启用和配置遥测输出。

其他资源

有关启用和使用可观测性的更多信息，请参阅以下资源：

- OpenTelemetry
- 博客文章《增强可观察性 适用于 .NET 的 AWS SDK》 OpenTelemetry
- 博客文章宣布 AWS .NET OpenTelemetry 库正式上市。
- 的出口商 OpenTelemetry

- 有关可观测性的示例 AWS Tools for PowerShell，请参阅《PowerShell 用户工具指南》中的可观察性。

配置 TelemetryProvider

您可以在应用程序中为所有服务客户端或单个客户端全局配置 TelemetryProvider，如以下示例所示。the section called “遥测提供商”一节提供了有关遥测实现的信息，包括有关随 SDK 提供的实现的信息。

配置默认的全局遥测提供程序

默认情况下，每个服务客户端都尝试使用全局可用的遥测提供程序。这样，您只需设置一次提供程序，所有客户端都会使用它。此操作只需执行一次，且必须在创建任何服务客户端之前进行。

以下代码片段展示了如何设置全局遥测提供程序。然后，它会创建一个 Amazon S3 服务客户端，并尝试执行失败的操作。该代码向应用程序添加追踪记录和指标。此代码使用以下 NuGet 软件包：OpenTelemetry.Exporter.Console和OpenTelemetry.Instrumentation.AWS。

> **Note**
>
> 如果您使用身份 AWS IAM Identity Center 验证，请务必同时添加AWSSDK.SSO和AWSSDK.SSOOIDC。

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

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

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

```
    .Build();

var s3Client = new AmazonS3Client();

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

Console.Read();
```

为特定服务客户端配置遥测提供程序

您可以为单个服务客户端配置特定的遥测提供程序（而不是全局遥测提供程序）。为此，请使用服务客户端构造函数的 Config 对象的 `TelemetryProvider` 类。例如，请参阅 AmazonS3Config 并找到 `TelemetryProvider` 属性。有关自定义遥测实现的信息，请参阅the section called “遥测提供商”。

主题

- Metrics
- 遥测提供商

Metrics

下表列出了 SDK 发出的遥测指标。配置遥测提供程序以使指标可观察。

发布了哪些指标？

指标名称	单位	类型	Attributes	描述
client.call.持续时间	s	直方图	rpc.service，rpc.m	总体呼叫时长（包括重试次数、发送或接收请求和响应正文的时间）
客户机正常运行时间	s	直方图	rpc. 服务	自创建客户机以来的时间长度

指标名称	单位	类型	Attributes	描述
client.call.tept	{尝试}	Monoton Counter	rpc.servi ce，rpc.m	单个操作的尝试次数
client.call.errors	{错误}	Monoton Counter	rpc.servi ce、rpc.me thod、excep	某项操作的错误数
client.call.tempt_ duration	s	直方图	rpc.servi ce，rpc.m	连接到服务、发送请求以及取回 HTTP 状态码和标头所花费的时间（包括等待发送的排队时间）
client.call.resolv e_endpoint_持续时间	s	直方图	rpc.servi ce，rpc.m	为请求解析终端节点（终端节点解析器，不是 DNS）所花费的时间
client.call.serial ization_dur	s	直方图	rpc.servi ce，rpc.m	序列化消息正文所花费的时间
client.call.deriza tion_duration_	s	直方图	rpc.servi ce，rpc.m	反序列化消息正文所花费的时间
client.col.auth.si gning_duration	s	直方图	rpc.servi ce，rpc.m	签署请求所花费的时间
client.col.auth.re solve_ide ntity_duration	s	直方图	rpc.servi ce，rpc.m	从身份提供商处获取身份（例如 AWS 凭证或持有者令牌）所花费的时间
client.http.bytes_ sent	方式	Monoton Counter	服务器地址	HTTP 客户端发送的总字节数
client.http.bytes_ receiv	方式	Monoton Counter	服务器地址	HTTP 客户端接收的总字节数

以下是各列的描述：

- 指标名称-发出的指标的名称。

- 单位-指标的计量单位。单位以 UC UM 区分大小写 (“c/s”) 表示法给出。
- 类型-用于捕获指标的仪器类型。
- 属性-与指标一起发出的一组属性（维度）。
- 描述-对指标所衡量内容的描述。

遥测提供商

SDK 提供了OpenTelemetry作为遥测提供商的实现，下一节将对此进行介绍。

如果您有特定的遥测要求，已经在考虑遥测解决方案，或者需要精细控制遥测数据的捕获和处理方式，则也可以实现自己的遥测提供程序。

在TelemetryProvider课堂上注册你自己的实现。以下是如何注册自己的`TracerProvider`和的简单示例`MeterProvider`。

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

public class CustomTracerProvider : TracerProvider
{
    // Implement custom tracing logic here
}
public class CustomMeterProvider : MeterProvider
{
    // Implement custom metrics logic here
}

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

主题

- 配置 OpenTelemetry基于遥测的提供商

配置 OpenTelemetry基于遥测的提供商

适用于 .NET 的 AWS SDK 包括 OpenTelemetry基于遥测提供程序的实现。有关如何将此提供程序设置为全局遥测提供程序的详细信息，请参阅配置 `TelemetryProvider`。要使用此遥测提供程序，您需要以下项目资源：

- .Instrum OpenTelemetryentation.aws NuGet 软件包。
- 遥测导出器，例如 OTLP 或控制台。有关更多信息，请参阅 OpenTelemetry文档中的导出器。

可以对 SDK 中包含的 OpenTelemetry 实现进行配置，以减少对 HTTPS 请求、凭证和压缩的跟踪量。为此，请将 `SuppressDownstreamInstrumentation` 选项设置为 `true`，类似于以下内容：

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

有关此提供商的更多信息，请参阅 with 中的博客文章 “ 适用于 .NET 的 AWS SDK 增强可观察性”。OpenTelemetry

有关用户和角色的其它信息

要在.NET 上进行.NET 开发 AWS 或在上 AWS运行.NET 应用程序，需要有适合这些任务的用户、权限集和服务角色的某种组合。

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，最佳做法也是在中创建用户和权限集 AWS IAM Identity Center 并使用身份源提供的临时证书。

对于开发，您可以使用自己创建的或在使用进行身份验证 AWS中提供的用户。如果您拥有适当的 AWS 管理控制台 权限，还可以为该用户创建权限最低的不同权限集，或者创建专门用于开发项目的新用户，提供权限最小的权限集。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，请参阅工具参考指南中的身份验证AWS SDKs和访问权限以及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 管理控制台 并打开 IAM 控制台，网址为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，查找并选择 EC2（例如），然后选择EC2用例（例如）。
4. 选择 “下一步：权限”，然后为应用程序将使用的 AWS 服务选择相应的策略。

   > **Warning**
   >
   > 请勿选择该AdministratorAccess策略，因为该策略允许您账户中几乎所有内容的读取和写入权限。

5. 选择下一步: 标签，然后输入所需的任何标签。

   您可以在 IAM 用户指南中使用 AWS 资源标签控制访问权限中找到有关标签的信息。
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。
- 对在 Amazon EC2 实例上运行的应用程序使用 IAM 角色。

适用于 .NET 的 AWS SDK 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 接口
- 配置其他应用程序参数
- 的配置文件参考 适用于 .NET 的 AWS SDK

使用 AWSSDK.Extensions.NETCore.Setup 和 iConfiguration 接口

（此主题以前的标题是“适用于 .NET 的 AWS SDK 使用 .NET Core 配置”）

.NET Core 最大的变化之一是删除了.NET 框架`app.config`和 ASP.NET 应用程序中使用的标准和`web.config`文件。`ConfigurationManager`

.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 适用于 .NET 的 AWS SDK 与 .NET Core 一起使用，您可以使用该AWSSDK.Extensions.NETCore.Setup NuGet 软件包。与许多 .NET Core 库一样，它在`IConfiguration`界面中添加了扩展方法，以实现无缝 AWS 配置。

此软件包的源代码 GitHub 位于https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup。

使用 AWSSDK.Extensions.NETCore.Setup

假设你创建了一个 ASP.NET 核心 Model-View-Controller (MVC) 应用程序，这可以通过在 Visual Studio 中使用ASP.NET 核心 Web 应用程序模板或在.NET Core CLI `dotnet new mvc ...` 中运行来完成。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;
}
```

要使用该Configuration对象获取AWS选项，请先添加AWSSDK.Extensions.NETCore.Setup NuGet 软件包。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。当您部署已EnvironmentName设置为生产的 Amazon EC2 实例时，该文件将被忽略，并将退回到为该 Amazon EC2 实例配置的 IAM 证书和区域。 适用于 .NET 的 AWS SDK

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>
```

要从代码访问文件中设置的 AWS 选项，请调用添加到的GetAWSOptions扩展方法IConfiguration。

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 包添加到您的项目中。）

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

- Region
- 配置文件
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse

- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

ASP.NET 核心依赖注入

该AWSSDK.Extensions.NETCore.Setup NuGet 软件包还集成了 ASP.NET Core中新的依赖注入系统。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。

该`AWSSDK.Extensions.NETCore.Setup` NuGet 软件包添加了新的扩展方法`IServiceCollection`，可用于向依赖注入添加 AWS 服务。以下代码向您展示如何添加读取`IConfiguration`的 AWS 选项，以将 Amazon S3 和 DynamoDB 添加到服务列表中。（请务必将AWSSDK.S3和AWSSDK.DynamoDBv2 NuGet 包添加到您的项目中。）

```
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;
    }

    ...

}
```

> **Note**
>
> 默认情况下，将服务客户端AddAWSService注册为单例，这是建议的生命周期，因为服务客户端是线程安全的，而且构造成本相对较高。有关重复使用客户端和其他性能建议的更多信息，请参阅性能最佳实践。

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 服务生成的终端节点

这些参数可以在应用程序的 App.config 或 Web.config 文件中配置。尽管您也可以使用 适用于 .NET 的 AWS SDK API 对其进行配置，但我们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，请参阅适用于 .NET 的 AWS SDK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。这是默认值。

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 适用于 .NET 的 AWS SDK API 合并LoggingOptions枚举值并设置 AWSConfigs.Logging 属性：

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

对此设置的更改仅对新的 AWS 客户端实例生效。

AWSLog指标

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">
```

或者，要使用 适用于 .NET 的 AWS SDK API 设置指标日志记录，请设置AWSConfigs。 LogMetrics财产:

```
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的 AWS 客户端实例生效。

AWSRegion

为未明确指定 AWS 区域的客户端配置默认区域。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 <appSettings> 部分中设置 AWSRegion 关键字：

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

或者，要使用 适用于 .NET 的 AWS SDK API 设置区域，请设置AWSConfigs。 AWSRegion财产:

```
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的 AWS 客户端实例生效。

AWSResponse正在记录

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。这是默认值。

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>
```

或者，在以下<appSettings>部分中AWSResponse设置 Logging 密钥：

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

或者，要使用 适用于 .NET 的 AWS SDK API 设置服务日志记录，请设置AWSConfigs。ResponseLogging将属性设置为ResponseLoggingOption枚举的其中一个值：

```
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 适用于 .NET 的 AWS SDK API 设置表名前缀，请设置 AWSConfigs.Dynamo 属性 DBContext TableNamePrefix：

```
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 适用于 .NET 的 AWS SDK API 设置签名版本 4 签名，请将 AWSConfigs.S3 UseSignatureVersion 4 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。

AWSEndpoint定义

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"/>
```

或者，你可以在以下<appSettings>部分中设置AWSEndpoint定义密钥：

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

或者，要使用 适用于 .NET 的 AWS SDK API 设置端点定义文件，请设置AWSConfigs。EndpointDefinition财产:

```
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的 AWS 客户端实例生效。该endpoints.json文件可从中获得https://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json。

AWS 服务生成的终端节点

有些 AWS 服务会生成自己的终端节点，而不是使用区域终端节点。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。这些服务的两个例子是 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 终端节点示例

亚马逊 CloudSearch 客户端用于访问亚马逊 CloudSearch 配置服务。您可以使用 Amazon CloudSearch 配置服务创建、配置和管理搜索域。要创建搜索域，请创建一个CreateDomainRequest对象并提供该DomainName属性。使用请求AmazonCloudSearchClient对象创建对象。调用 CreateDomain 方法。调用返回的CreateDomainResponse对象包含一个同时具有DocService和SearchService端点的DomainStatus属性。创建一个AmazonCloudSearchDomainConfig对象并使用它来初始化该AmazonCloudSearchDomainClient类的SearchService实例DocService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 端点示例

要获取的终端节点 AWS IoT，请创建一个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，然后使用该对象实例化该类。AmazonIotDataClient

```
string iotEndpointAddress;
using (var iotClient = new AmazonIoTClient())
{
    var endPointResponse = iotClient.DescribeEndpoint();
    iotEndpointAddress = endPointResponse.EndpointAddress;
}
```

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

的配置文件参考 适用于 .NET 的 AWS SDK

> **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App.config或Web.config文件来指定 AWS 设置，例如 AWS 证书、日志选项、 AWS 服务终端节点和 AWS 区域，以及一些 AWS 服务的设置，例如亚马逊 DynamoDB、Amazon 和 Ama EC2 zon 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**
>
> 尽管您可以继续使用App.config或Web.config文件中的<appSettings>元素来指定 AWS 设置，但我们建议您按照本主题后面的说明使用<configSections>和<aws>元素。有关该<appSettings>元素的更多信息，请参阅配置 适用于 .NET 的 AWS SDK 应用程序中的<appSettings>元素示例。

> **Note**
>
> 尽管您可以继续在代码文件中使用以下AWSConfigs类属性来指定 AWS 设置，但以下属性已过时，将来的版本可能不支持这些属性：
>
> - DynamoDBContextTableNamePrefix

- EC2UseSignatureVersion4
- LoggingOptions
- LogMetrics
- ResponseLoggingOption
- S3UseSignatureVersion4

通常，我们建议您不要在代码文件中使用AWSConfigs类属性来指定 AWS 设置，而应使用App.config或Web.config文件中的<configSections>和<aws>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 适用于 .NET 的 AWS SDK 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

您可以从<aws>元素内部在App.config或Web.config文件中指定 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 编辑器不为<aws>元素或其子元素提供自动代码完成功能 (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

以下是 AWS 设置部分中允许使用的元素的列表。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
- 发电机 DBContext
- ec2
- logging
- 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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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

该<aws>元素代表 AWS 设置部分中最上面的元素。此元素可以包含以下属性：

endpointDefinition

定义要使用的 AWS 区域和端点的自定义配置文件的绝对路径。此属性映射到 适用于 .NET 的 AWS SDK中的 Amazon.AWSConfigs.EndpointDefinition 属性。

profileName

用于进行服务调用的存储 AWS 凭据的配置文件名称。此属性映射到 适用于 .NET 的 AWS SDK中的 Amazon.AWSConfigs.AWSProfileName 属性。

profilesLocation

指向与其他人共享的凭据文件位置的绝对路径 AWS SDKs。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 适用于 .NET 的 AWS SDK中的 Amazon.AWSConfigs.AWSProfilesLocation 属性。

region

未明确指定 AWS 区域的客户端的默认区域 ID。此属性映射到 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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>
```

发电机 DBContext

<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，该属性表示 DynamoDB 上下文在未手动配置的情况下将使用的默认表名前缀。此属性从 适用于 .NET 的 AWS SDK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

该<ec2>元素表示 Amazon EC2 设置的集合。此元素可以包含 useSignatureVersion4 属性，该属性指定是否将对所有请求使用签名版本 4 签名（true），或者是否不对所有请求使用签名版本 4 签名（false，默认值）。此属性从 适用于 .NET 的 AWS SDK 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" />
```

logging

<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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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

记录服务响应的时间（从 适用于 .NET 的 AWS SDK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

登录到哪里（从中的LogToAmazon.AWSConfigs.LoggingConfig.LogTo属性映射到该属性 适用于 .NET 的 AWS SDK）。

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

该<map>元素表示从.NET 类型到 DynamoDB 表的 type-to-table映射集合中的单个项目（映射到中Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings属性的TypeMapping类实例）。 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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。（此元素映射到 Amazon.Util 的实例。 PropertyConfig 中的AddProperty方法中的类 适用于 .NET 的 AWS SDK）有关更多信息，请参阅 DynamoDB 开发工具包和 D ynamoDB 属性的增强功能。

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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK中的 Amazon.Util.PropertyConfig.Ignore 属性。

name

属性的名称。此属性映射到 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK 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 适用于 .NET 的 AWS SDK中的 Amazon.AWSConfigs.ProxyConfig.Host 属性映射到 Amazon.Util.ProxyConfig.Host 属性。

password

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

端口

代理的端口号。此属性从 适用于 .NET 的 AWS SDK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 适用于 .NET 的 AWS SDK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，该属性指定是否将对所有请求使用签名版本 4 签名（true），或者是否不对所有请求使用签名版本 4 签名（false，默认值）。此属性映射到 适用于 .NET 的 AWS SDK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者的联合身份验证，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 SDKs 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，如中所述使用进行身份验证 AWS。

有关凭证的重要警告和指南

有关凭证的警告

- 请勿使用您账户的根凭证访问 AWS 资源。这些凭证可提供不受限的账户访问且难以撤销。
- 不得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，存储在共享 AWS `credentials`文件中的任何凭据都以纯文本形式存储。

有关安全管理凭证的更多指南

有关如何安全管理 AWS 证书的一般性讨论，请参阅 IAM 用户指南中的AWS 安全证书AWS 一般参考和《IAM 用户指南》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 凭据文件
- 使用 SDK Store（仅适用于 Windows）

使用共享 AWS 凭据文件

（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。这个文件可以被另一个人使用 AWS SDKs。V isual Studio AWS Tools for Windows PowerShell、JetBrains、和 VS Code 的 AWS 工具包也可以使用它。AWS CLI

> **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者的联合身份验证，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 SDKs 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，如中所述使用进行身份验证 AWS。

一般信息

默认情况下，共享 AWS 凭据文件位于您的主`.aws`目录中的目录中，其名称为`credentials`：`~/.aws/credentials`（Linux 或 macOS）或`%USERPROFILE%\.aws\credentials`（Windows）。有关其他位置的信息，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的共享文件位置。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 SDKs 和工具参考指南》中的凭证文件格式。

您可以通过多种方式管理共享 AWS 凭据文件中的配置文件。

- 使用任何文本编辑器创建和更新共享 AWS 凭据文件。
- 使用 Amazon.Runtime。 CredentialManagement 适用于 .NET 的 AWS SDK API 的命名空间，如本主题后面所示。
- 使用适用于 V isual Studio AWS Tools for PowerShell和 VS Code 的 AWS 工具包的命令和过程。JetBrains
- 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

以下各节显示共享 AWS 凭据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。其它示例显展示了如何使用特定方法。

默认配置文件

共享 AWS 凭据文件几乎总是会有一个名为 defa ult 的配置文件。如果未定义其他配置文件，则在此处适用于 .NET 的 AWS SDK 查找凭据。

`[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

此示例向您展示如何创建配置文件并以编程方式将其保存到共享 AWS 凭据文件中。它使用 Amazon.Runtime 的以下类。 CredentialManagement命名空间：CredentialProfileOptionsCredentialProfile、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 的以下类。CredentialManagement命名空间：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**
>
> 您也可以使用其他方法在其他位置设置 AWS 区域。有关更多信息，请参阅 为设置 AWS 区域 适用于 .NET 的 AWS SDK。

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 商店是另一个为 适用于 .NET 的 AWS SDK 应用程序创建配置文件和存储加密凭据的地方。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者的联合身份验证，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 SDKs 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，如中所述使用进行身份验证 AWS。

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 Amazon.Runtime。 CredentialManagement 适用于 .NET 的 AWS SDK API 的命名空间，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 的以下类。 CredentialManagement命名空间:CredentialProfileOptionsCredentialProfile、和 Net SDKCredentials File。

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

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

void WriteProfile(string profileName, string keyId, string secret)
{
```

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

> **Warning**
>
> 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 的以下类。CredentialManagement命名空间：CredentialProfile和网络SDKCredentials文件。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**
>
> 您也可以使用其他方法在其他位置设置 AWS 区域。有关更多信息，请参阅 为设置 AWS 区域 适用于 .NET 的 AWS SDK。

使用 适用于 .NET 的 AWS SDK

本节提供有关在创建应用程序时 适用于 .NET 的 AWS SDK 可能需要考虑的功能的信息。

在开始之前，请务必设置环境并配置项目。

有关为特定 AWS 服务开发软件的信息以及代码示例，请参阅呼叫 AWS 服务。有关其他代码示例，请参阅 适用于 .NET 的 SDK (v4) 代码示例。

主题

- 使用异步编程 适用于 .NET 的 AWS SDK
- 的性能最佳实践 适用于 .NET 的 AWS SDK
- 在中使用分页结果 适用于 .NET 的 AWS SDK
- 对 HTTP 2 的支持 适用于 .NET 的 AWS SDK
- 其他工具

使用异步编程 适用于 .NET 的 AWS SDK

适用于 .NET 的 AWS SDK 使用Task-based 异步模式 (TAP) 实现其异步实现。要了解有关 TAP 的更多信息，请参阅 docs.microsoft.com 上的Task-based 异步模式 (TAP)。

本主题概述了如何在呼叫 AWS 服务客户端时使用 TAP。

适用于 .NET 的 AWS SDK 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。

> **Warning**
>
> 阻塞使用ResultWait()、或GetAwaiter().GetResult()是异步同步模式的异步调用。只有在没有控制台的简单控制台Main中才能接受SynchronizationContext。在ASP.NET、Windows FormsWPF、和中Blazor WebAssembly，这种模式可能会导致线程池枯竭或死锁。任何一种结果都会导致您的应用程序挂起。最好声明Main为，async然后使用await，如第一个示例所示。有关更多信息，请参阅 async/await 正确使用。

您可以在（简单跨平台应用程序和基于 Windows 的简单应用程序）创建简单的应用程序部分和中查看异步调用 AWS 服务客户端的完整示例引导式代码示例。

的性能最佳实践 适用于 .NET 的 AWS SDK

如何创建客户端、处理响应和配置应用程序对吞吐量、延迟和内存使用有很大影响。本主题介绍可帮助您的应用程序高效、可靠地运行的使用和配置模式。这些模式在高负载或资源受限的环境（例如容器和无服务器函数）中最为重要。遵循这些做法可以提高性能并防止常见问题，例如响应缓慢、挂起和内存使用率过高。

最有影响力的做法如下：

- 重复使用单个长期服务客户端，而不是为每个请求创建一个服务客户端。
- 处置响应和流，以便将网络连接释放回连接池。
- `await`正确使用`async`/，切勿阻塞异步 SDK 调用。
- ???为受限环境（例如 AWS Lambda 和 Amazon ECS）配置.NET 垃圾回收功能。
- 在高吞吐量下管理 HTTP 连接和连接限制。

在开始之前，请务必设置环境并配置项目。

重复使用单个长期服务客户端

诸如 Amazons3Client之类的服务客户端AmazonDynamoDBClient是线程安全的，构造成本相对较高且使用寿命长。 构建客户端可以解析区域和终端节点信息，并建立底层 HTTP 基础架构。为每项服务创建一个客户端，并在应用程序的整个生命周期中重复使用。如果您致电多个区域，请为每个区域创建一个单独的客户端。

> **Warning**
>
> 不要为每个请求或在循环内创建新的服务客户端。反复创建客户端会破坏底层 HTTP 基础架构，从而增加可衡量的延迟。它还会耗尽插座或手柄，导致请求失败或在负载下挂起。

在使用依赖注入的应用程序中，将客户端注册为单例。`AWSSDK.Extensions.NETCore.SetupNuGet`包中的`AddAWSService`扩展方法将客户端注册的默认生命周期为`ServiceLifetime.Singleton`。客户端是在第一次请求时创建的，并且在进程的

整个生命周期中都会重复使用相同的实例。有关使用依赖注入注册 AWS 服务以及从配置中读取选项的更多信息，请参阅AWSSDK.Extensions.NETCore.Setup 和 iconfiguration。

您也可以手动将客户端注册为单例。

```
builder.Services.AddSingleton<IAmazonS3>(_ => new AmazonS3Client());
```

> **Note**
>
> 因为默认情况下会将客户端AddAWSService注册为单例，所以不要处置它提供的客户端。如果您需要非默认生命周期，请将不同的ServiceLifetime值传递给的可选lifetime参数。AddAWSService

重复使用客户端并不意味着你应该停止处理每个操作的响应。在应用程序生命周期内重复使用客户端，但继续处置各个操作返回的响应和流，如中所述处置响应和直播。

处置响应和直播以释放连接

一些 SDK 响应对象带有网络直播流。最常见的示例是 GetObjectResponse，它通过其ResponseStream属性实现IDisposable和公开对象的内容。在完全读取流或处理响应之前，响应会保持开放的 HTTP 连接。如果您泄露了这些响应（例如，通过循环调用GetObject而不处置每个结果），则打开的连接会累积直到连接池耗尽，下一次调用就会阻塞。这是关于第 N 个对象上的下载 “随机挂起” 或停滞的报道背后的原因。

务必将直播响应打包在using声明中，并立即阅读或复制其直播。

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

// s3Client is a reused, long-lived client.
var request = new GetObjectRequest { BucketName = bucketName, Key = key };

using var response = await s3Client.GetObjectAsync(request);
await response.WriteResponseStreamToFileAsync(filePath, append: false,
 CancellationToken.None);
```

处置响应和处置响应ResponseStream是等效的；任何一个都关闭底层网络流并将连接返回到池。

Tip

如果您只需要对象元数据，例如大小、上次修改时间、内容类型或 ETag，请调用`GetObjectMetadata`或`GetObjectMetadataAsync`代替。`GetObject`元数据操作发出 HTTP HEAD 请求且不传输任何对象正文，因此无需管理任何内容流。

```
var metadata = await s3Client.GetObjectMetadataAsync(bucketName, key);
Console.WriteLine($"Size: {metadata.ContentLength} bytes");
```

Warning

响应和流类型（例如）`GetObjectResponse`不实现终结器，因此您不能依靠垃圾回收来为您释放它们的连接。您必须使用`using`或显式`Dispose`调用确定性地处理响应和流。

async/await 正确使用

在现代 .NET 上，中的服务操作 适用于 .NET 的 AWS SDK 是异步的，返回 a `Task`。在 .NET Framework 上，同步方法也存在，但异步调用的扩展性更好，建议使用。使用代码执行操作`await`，并`async`传播到代码的每一层，直至入口点。有关使用 SDK 进行异步编程的更多信息，请参阅异步编程。

Warning

不要使用`.Result`、`.Wait()`或屏蔽异步 SDK 调用`.GetAwaiter().GetResult()`。这种异步同步模式是应用程序挂起的常见原因：

- 在负载下，阻塞调用消耗线程的速度快于线程池的增长速度，因此无法连续运行。这种线程池匮乏表现为无限期挂起，是现代 .NET（ASP.NET Core没有默认值）上的主要故障模式。`SynchronizationContext`
- 某些上下文捕获`SynchronizationContext`，例如经典ASP.NET、Windows FormsWPFBlazor WebAssembly、和.NET 框架应用程序。在这些情况下，阻塞调用线程而延续需要相同的线程会导致死锁。SDK 中的操作在`ConfigureAwait(false)`内部使用，因此它们不会将其延续内容发布回捕获的上下文中。死锁是由调用链中其他地方的`async`代码造成的，这些代码确实捕获了死锁。自`await`始至终使用SDK调用可以完全避免问题。

以下方法阻塞异步调用，可能导致线程池死锁或耗尽。

```
// Anti-pattern: do not do this.
public GetObjectResponse Get(GetObjectRequest request)
{
    return s3Client.GetObjectAsync(request).Result;
}
```

相反，进行方法async和await调用。

```
public async Task<GetObjectResponse> GetAsync(GetObjectRequest request)
{
    return await s3Client.GetObjectAsync(request);
}
```

异步代码的其他建议：

- 将一个传递CancellationToken给每个操作，这样慢速或停顿的呼叫就可以取消，而不是挂起。有关更多信息，请参阅 使用 CancellationToken 参数配置超时。
- 不要在空catch块中吞下异常。这样做会掩盖真正的失败，使挂机与错误无法区分。捕获特定的异常并记录它们。
- 重新抛出捕获的异常时，使用throw;而不是 s throw ex; o 来保留原始堆栈跟踪。
- 如果您必须从同步边界调用，则将其视为最后的手段，并将工作与捕获的上下文隔离开来，而不是将阻塞调用设为默认模式。

为配置 .NET 垃圾收集 AWS Lambda 还有亚马逊 ECS

当容器出现 “内存泄漏” 时，.NET 垃圾收集器 (GC) 可能会保留回收的内存以供重复使用。因此，即使托管堆没有增长，进程内存也可能显得很高且稳定。此外，GC 不会自动检测容器的内存限制（其 cgroup 限制）。在受限的环境中，堆可能会向主机的内存而不是容器的极限增长。这可能导致OutOfMemoryException或容器被终止。

为了帮助 GC 在受限的环境中正常运行：

- 在容器上设置明确的内存限制，以便 GC 遵守 cgroup 限制， and/or 设置DOTNET_GCHeapHardLimit（绝对字节值，十六进制）或DOTNET_GCHeapHardLimitPercent环境变量来限制托管堆的上限。在报告的几个 Amazon ECS 内存不足案例中，设置硬内存限制解决了崩溃问题。

- 在小型 AWS Lambda 和 Amazon ECS 主机上，考虑禁用并发（后台）垃圾回收，这样收集器就不会预留额外内存；例如，将`DOTNET_gcConcurrent`环境变量设置为`0`，或`<ConcurrentGarbageCollection>false</ConcurrentGarbageCollection>`在项目文件中设置。
- 绑定您的并发量。同时启动许多操作，例如调用`Task.WhenAll`大量集合，会膨胀进程内存，并可能耗尽连接池。改为限制并行度。例如，避免这种无界模式：

```
// Anti-pattern: starts one task per item with no limit.
await Task.WhenAll(keys.Select(key => s3Client.GetObjectMetadataAsync(bucket, key)));
```

取而代之的是，使用`Parallel.ForEachAsync`以下方法限制并发量：

```
var options = new ParallelOptions { MaxDegreeOfParallelism = 10 };
await Parallel.ForEachAsync(keys, options, async (key, token) =>
{
    await s3Client.GetObjectMetadataAsync(bucket, key, token);
});
```

有关这些设置的更多信息，请参阅 learn.microsoft.com 上的垃圾收集运行时配置选项。有关 AWS 计算的特定指导，请参阅 AWS 开发者工具博客文章为 Amazon ECS 配置.NET 垃圾回收和 AWS Lambda。

管理 HTTP 连接和连接限制

在高吞吐量下，两个与连接相关的问题很常见。首先是打开了太多短暂的连接。这会耗尽临时端口，留下套接字，并增加 TCP 和 TLS 握手延迟。`TIME_WAIT`第二个问题是可用的连接太少，这给并行性带来了瓶颈。重复使用单个长寿命客户端（参见重复使用服务客户端）是健康连接池的基础，因为池化取决于客户端的重复使用。

要调整每个端点的并发连接数，请在客户端配置上设置该`MaxConnectionsPerServer`属性。当此属性为`null`（默认值）时，基础`HttpClientHandler`默认值适用，这在现代 .NET 中实际上是无限的。仅当对同一端点的多个并发请求出现连接瓶颈时才提高。一个不错的起点是每个终端节点的预期并发请求的峰值数。在不提高吞吐量的情况下，将其设置得远高于工作负载需求会浪费套接字。

```
using Amazon.S3;

var config = new AmazonS3Config
{
```

```
    MaxConnectionsPerServer = 50
};

var s3Client = new AmazonS3Client(config);
```

如果您已经使用依赖注入，请通过配置客户端AWSSDK.Extensions.NETCore.Setup。当您使用 DI 或注册多个服务客户端时，推荐使用这种方法。它可以集中配置，使客户端易于注入和测试。您可以从应用程序的配置中设置配置值，而不是在代码中设置配置值。有关更多信息，请参阅AWSSDK.Extensions.NETCore.Setup 和 iconfiguration。

最后，绑定自己的并行度，这样启动的并发操作就不会超过连接限制允许的次数。例如，根据您的连接限制SemaphoreSlim进行门禁呼叫：

```
var throttle = new SemaphoreSlim(50); // match MaxConnectionsPerServer
await throttle.WaitAsync(token);
try
{
    await s3Client.GetObjectAsync(request, token);
}
finally
{
    throttle.Release();
}
```

配置超时和重试

超时和重试直接影响感知性能。Timeout值过高会使停滞的请求长时间阻塞。当服务已经返回限制错误时，激进的重试策略会增加更多请求，并可能使限制变得更糟。选择符合应用程序对延迟和失败容忍度的重试策略，让真正的异常传播，而不是以掩盖挂起的方式重试。

> **Note**
>
> 该Timeout属性不影响异步调用。如果您使用异步调用，请使用 CancellationToken 参数配置超时改为参见。

有关重试模式和Timeout属性（仅ReadWriteTimeout适用于.NET Framework）的更多信息以及如何设置它们的示例，请参阅重试和超时。MaxErrorRetry

优化串流和大对象传输 (Amazon S3)

要上传和下载大型对象或许多对象，请使用Amazon.S3.Transfer命名空间中的TransferUtility类。它使用分段传输并行上传和下载，并为您管理流、分段和连接。这比单流传输更快，也是移动大型对象的推荐方式。

- 并行分段下载。SDK 的早期版本以单个流的形式下载对象，而不是并行下载各个部分。从AWSSDK.S3版本 4.0.17 开始，通过、和方法TransferUtility提供分段（并行）下载。DownloadWithResponseAsync OpenStreamWithResponseAsync DownloadDirectoryWithResponseAsync使用时OpenStreamWithResponseAsync，当您消耗返回的流时，对象的各个部分会缓冲在内存中。使用的MaxInMemoryParts属性控制缓冲了多少个部件。 TransferUtilityOpenStreamRequest 对于大多数转账，首选TransferUtility。仅在需要特定范围或自定义并行GetObjectRequest模式时，才使用ByteRange属性的方式自行下载字节范围。有关更多信息，请参阅 AWS 开发者工具博客上介绍适用于.NET Transfer Manager 的 AWS SDK 的分段下载支持。
- 上传的内容长度。Amazon S3 PUT 需要已知的内容长度，默认情况下，开发工具包会在请求正文中计算校验和。当已知长度且可搜索流时，SDK 无需在内存中缓冲整个对象即可执行此操作。通过根据需要进行缓冲来为您TransferUtility处理不可搜索的流。如果您改为使用不可搜索的流（例如中的原始请求正文ASP.NET Core）PutObjectAsync直接调用，则请求可能会失败。提供可搜索的直播，明确设置内容长度，或配置 SDK 计算校验和的方式。有关更多信息，请参阅《软件开发工具包和 AWS 工具参考指南》中的数据完整性保护。
- 零件尺寸。Amazon S3 允许每次分段上传最多 10,000 个分段。当总长度已知时，SDK 会自动计算出保持在该限制范围内的分段大小。你主要需要为PartSize事先不知道长度的直播做好准备。否则，较小的默认分段可能会超过超大上传的限制。更大的部件大小还可以减少每个部件的开销，但代价是每个部件需要更多的内存。

诊断性能问题

当您调查减速、挂起或明显泄漏时，以下信号可帮助您快速找到原因：

- 越来越多的TIME_WAIT处于CLOSE_WAIT或状态（可见netstat）的套接字是未处理的响应或每个请求创建和丢弃的客户端的指纹。请参阅处置响应和直播和重复使用服务客户端。
- 启用请求指标和响应日志，以确认请求实际正在分发并测量延迟。在创建服务客户端之前，在 AWSConfigs 上设置LoggingConfig对象的属性。客户端捕获设置，例如LogMetrics在构造时捕获设置。

```
using Amazon;
```

```
AWSConfigs.LoggingConfig.LogMetrics = true;
AWSConfigs.LoggingConfig.LogResponses = ResponseLoggingOption.OnError;
```

- 研究内存时，将全进程内存与托管堆区分开。高但稳定的进程内存通常是 GC 保存回收的内存，而不是泄漏。请参阅配置垃圾收集。

在中使用分页结果 适用于 .NET 的 AWS SDK

某些 AWS 服务收集和存储大量数据，您可以使用的 API 调用来检索这些数据 适用于 .NET 的 AWS SDK。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适用于 .NET 的 AWS SDK 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，则会看到一个`Paginators`属性。该属性为亚马逊 CloudWatch 日志提供分页器。

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`面提到的内容中，该`Paginator`对象包含一个`Responses`属性，其中的完整DescribeLogGroupsResponse对象来自 API 调用。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 框架 4.7.2（或更高版本）项目中提供同步分页。异步分页可在 .NET Core 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。有关如何使用同步分页和 .NET Framework 4.7.2（或更高版本）执行相同任务的信息显示在中的示例之后。分页工具的其它注意事项

示例

以下示例向您展示如何使用显示日志组列表。 适用于 .NET 的 AWS SDK 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。

在没有分页器的情况下获取 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);
      }
```

这两个片段的结果完全相同，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。
> 您可能还需要该Microsoft.Bcl.AsyncInterfaces NuGet 软件包，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

  课堂 AmazonCloudWatchLogsClient
- 命名空间 Amazon.CloudWatchLogs.Model

  课堂 DescribeLogGroupsRequest

  课堂 DescribeLogGroupsResponse

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

分页工具的其它注意事项

- 分页工具不能多次使用

如果您需要在代码的多个位置 AWS 显示特定分页器的结果，则不得多次使用分页器对象。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

  同步分页可用于 .NET 框架 4.7.2（或更高版本）项目。

  要查看此信息，请创建一个 .NET Framework 4.7.2（或更高版本）项目并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。

对 HTTP 2 的支持 适用于 .NET 的 AWS SDK

某些 AWS 服务和操作需要 HTTP 2。例如，无法通过 HTTP 1.1 在 Amazon Transcribe Streaming 中进行双向流式传输，因此需要改用 HTTP 2。版本 4 适用于 .NET 的 AWS SDK 增加了对 HTTP 2 的支持，因此您可以在应用程序中使用这些操作。对于双向 HTTP 2 操作，SDK 在接收直播时的行为与 HTTP 1.1 的行为类似。也就是说，当使用 SDK 的应用程序向服务发送事件时，该请求会有一个由开发者分配的发布者。

要了解这种行为的实际效果，请考虑以下 Amazon Transcribe Streaming 示例。它使用亚马逊。TranscribeStreaming还有亚马逊。 TranscribeStreaming.模型命名空间。

在此示例中，开发人员使用回调函数（.NET）定义该`StartStreamTranscriptionRequest.AudioStreamPublisher`属性`Func`。SDK 使用`Func`定义的 f `AudioStreamPublisher` or 从用户的代码中提取事件以流式传输给用户。SDK 会调用，`Func`直到它返回空值。

该代码演示了如何将文件中的音频流式传输到 Amazon Transcribe Streaming 中进行处理。

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

```
using Amazon.TranscribeStreaming.Model;

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

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

Stream fileStream = File.OpenRead("hello-world.flac");
var buffer = new byte[1024 * 10];
startRequest.AudioStreamPublisher += async () =>
{
    var bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);

    if (bytesRead == 0)
        return null;

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

    return audioEvent;
};

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

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

void TranscriptResultStream_ExceptionReceived(object? sender,
 Amazon.Runtime.EventStreams.EventStreamExceptionReceivedArgs<TranscribeStreamingEventStreamExc
 e)
{
```

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

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

_ = response.TranscriptResultStream.StartProcessingAsync();

try
{
    await Task.Delay(10000, cancelSource.Token);
}
catch (TaskCanceledException) { }
```

Warning

一些双向 HTTP 2 操作，例如来自亚马逊 Bedrock InvokeModelWithBidirectionalStreamAsync 的方法，即亚马逊。 BedrockRuntime命名空间，在某些事件发布之前，不要在服务客户端上返回初始调用的响应。此行为可能会导致您的应用程序被阻止。为避免这种情况，请将向发布者提供事件的应用程序代码分开，然后在与在服务客户端上调用操作的线程不同的线程上运行该代码。

额外注意事项

- 适用于 .NET 的 AWS SDK 仅在以.NET 8 及更高版本为目标的版本中才支持 HTTP 2。它在以.NET 框架为目标的版本中不可用。

- 有关更多详细信息，请参阅aws-sdk-net GitHub存储库中的 PR 3730。

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 .NET 的消息处理框架

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 .NET 的消息处理框架。

与.NET Aspire 的集成

你可以利用与.NET Aspire 的集成来改善内部开发循环。有关更多信息，请参阅 在中 AWS 与.NET Aspire 集成 适用于 .NET 的 AWS SDK。

使用高级身份验证和授权 适用于 .NET 的 AWS SDK

本节中的主题提供有关 适用于 .NET 的 AWS SDK 应用程序中身份验证和授权的高级技术的信息。

主题

- 使用单点登录 适用于 .NET 的 AWS SDK

使用单点登录 适用于 .NET 的 AWS SDK

AWS IAM Identity Center 是一项基于云的单点登录 (SSO) 服务，可让您轻松集中管理对所有应用程序 AWS 账户 和云应用程序的 SSO 访问权限。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

简而言之，以类似于以下模式的方式与 IAM 身份中心 SDKs 进行交互：

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) 权限的短期证书。这种登录可以通过非 SDK 工具（如的）启动 AWS CLI，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。它提供了比使用进行身份验证 AWS中基本 SSO 设置更高级的补充信息。如果您不熟悉 SSO AWS，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 的实际运行情况：

- 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

在相关内容中配置 IAM Identity Center 后 AWS 账户，必须将 SSO 的命名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 访问入口。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

其中 AWS 区域 包含访问门户主机。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 及其代码的完整列表，请参阅中的区域终端节点Amazon Web Services 一般参考。

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。

- 使用生成令牌， AWS CLI 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 软件包，这样 SSO 解析才能起作用：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 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 软件包，这样 SSO 解析才能起作用：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未能引用这些程序包将导致运行时系统异常。

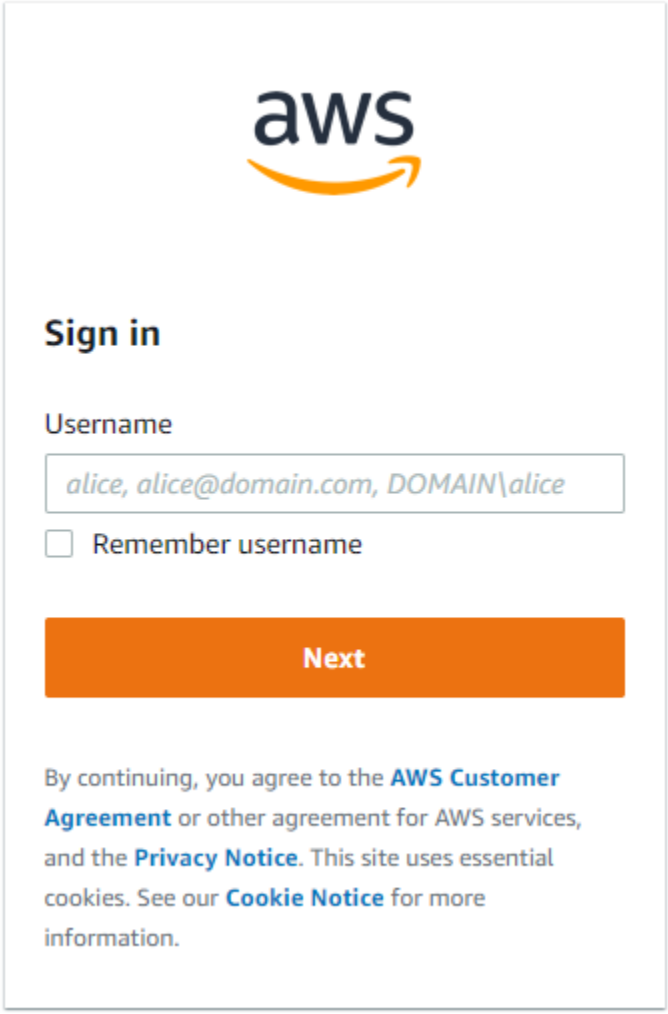
```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

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



> **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

本节介绍如何使用生成临时 SSO 令牌 AWS CLI，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]。

使用生成 SSO 令牌 AWS CLI

除了以编程方式生成临时 SSO 令牌外，您还可以使用生成 AWS CLI 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 软件包，这样 SSO 解析才能起作用：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 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？
- 配置 AWS CLI 为使用 IAM 身份中心
- 在中使用 IAM 身份中心证书 AWS Toolkit for Visual Studio

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。

本教程向您展示了中一小部分 SSO 功能。 适用于 .NET 的 AWS SDK有关将 IAM Identity Center 与一起使用的完整详细信息 适用于 .NET 的 AWS SDK，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可帮助您配置服务，例如 AWS Organizations 和 IAM Identity Center。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）。在安装和配置您的工具链 适用于 .NET 的 AWS SDK和入门之类的章节中对此进行了描述。
- 确定或创建至少一个 AWS 账户 可用于测试 SSO 的选项。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

本节介绍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 AWS 账户 身份登录测试。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. 在的横幅中 AWS 管理控制台，查找启用 SSO 时设置的。 AWS 区域 这是 AWS 账户 身份证左侧的下拉菜单。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：

      JSON

```
{
    "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

除了AWSSDK.SSO和之外AWSSDK.SSOOIDC，还包括 NuGet 软件包AWSSDK.SecurityToken。AWSSDK.S3

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


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

```
namespace SSOExample.S3.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

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

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

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

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

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

            var ssoCredentials = credentials as SSOAWSCredentials;
```

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

            return ssoCredentials;
        }

    }

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

列出 IAM 用户

除了AWSSDK.SSO和之外AWSSDK.SSOOIDC，还包括 NuGet 软件包AWSSDK.SecurityToken。AWSSDK.IdentityManagement

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

```
using System.Diagnostics;


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

namespace SSOExample.IAM.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

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

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

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

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

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

           var ssoCredentials = credentials as SSOAWSCredentials;

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

           return ssoCredentials;
       }

   }

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

```
    }
}
```

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对象的 Options 属性中提供回调方法来执行 SSO 登录任务。SSOAWSCredentials

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。

清理

如果您不想保留在本教程中创建的资源，请将其清理干净。这些可能是开发环境中的 AWS 资源或资源，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。它使用生成 AWS CLI 临时 SSO 令牌，而不是以编程方式生成。

本教程向您展示了中一小部分 SSO 功能。 适用于 .NET 的 AWS SDK有关将 IAM Identity Center 与一起使用的完整详细信息 适用于 .NET 的 AWS SDK，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可帮助您配置服务，例如 AWS Organizations 和 IAM Identity Center。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）。在安装和配置您的工具链 适用于 .NET 的 AWS SDK和入门之类的章节中对此进行了描述。
- 确定或创建至少一个 AWS 账户 可用于测试 SSO 的选项。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

本节介绍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 AWS 账户 身份登录测试。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. 在的横幅中 AWS 管理控制台，查找启用 SSO 时设置的。 AWS 区域 这是 AWS 账户 身份证左侧的下拉菜单。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：

JSON

```
{
    "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

除了AWSSDK.SSO和之外AWSSDK.SSOOIDC，还包括 NuGet 软件包AWSSDK.SecurityToken。AWSSDK.S3

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

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

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

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

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

```
            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

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

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

列出 IAM 用户

除了AWSSDK.SSO和之外AWSSDK.SSOOIDC，还包括 NuGet 软件包AWSSDK.SecurityToken。AWSSDK.IdentityManagement

```
using System;
```

```
using System.Threading.Tasks;

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

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

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

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

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

```
            }
            Console.WriteLine();
        }

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

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

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。

清理

如果您不想保留在本教程中创建的资源，请将其清理干净。这些可能是开发环境中的 AWS 资源或资源，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。你可以使用 AWS 管理控制台 或某些服务（如 CloudFormation 或）来做到这一点 AWS Cloud Development Kit (AWS CDK)。您也可以使用为部署目的而创建的 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.NET CLI AWS 工具将应用程序部署到 AWS：

- AWS 适用于.NET CLI 的部署工具-支持部署到AWS App Runner亚马逊弹性容器服务 (Amazon ECS) 和。AWS Elastic Beanstalk
- AWS Lambda 适用于.NET CLI 的工具-支持 AWS Lambda 项目部署。

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 的 AWS 部署工具 CLI 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

使用案例

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 — 该工具构建部署工件，使用生成的 AWS CDK 部署项目配置基础架构，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用例。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。
- 帮助学习 AWS CDK .NET-该工具可帮助您逐步学习它所依据的底层 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：

- Amazon ECS 服务使用 AWS Fargate-支持将 Web 应用程序部署到亚马逊弹性容器服务 (Amazon ECS)，计算能力由无服务器计算引 AWS Fargate 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。 APIs 无需事先具备基础设施经验。
- AWS Elastic Beanstalk-支持部署到可让开发人员轻松部署 Web 应用程序的服务以及大规模部署 APIs 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 — 该工具构建部署工件，使用生成的 AWS CDK 部署项目配置基础架构，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用例。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。
- 帮助学习 AWS CDK .NET-该工具可帮助您逐步学习它所依据的底层 AWS 工具，例如 AWS CDK.

AWS 部署工具支持将.NET 控制台应用程序部署到以下 AWS 服务：

- Amazon ECS 服务使用 AWS Fargate-支持将.NET 应用程序作为服务（例如后台处理器）部署到亚马逊弹性容器服务 (Amazon ECS)，计算能力 AWS Fargate 由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 — 该工具构建部署工件，使用生成的 AWS CDK 部署项目配置基础架构，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用例。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。
- 帮助学习 AWS CDK .NET-该工具可帮助您逐步学习它所依据的底层 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，允许您在不预置或管理服务器的情况下运行代码。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，请参阅什么是 L AWS ambda？在AWS Lambda 开发者指南中。

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，可以https://dotnet.microsoft.com/download去安装它。
- 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

以下说明假设您已经创建了 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. 出现提示时，输入 AWS 区域（您的 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 会自动为您监控 Lambda 函数，并通过亚马逊报告指标。 CloudWatch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。

为了帮助您的函数在 Lambda 的内存受限环境中高效运行，包括如何配置.NET 垃圾收集，请参阅。性能最佳实践

从中呼叫 AWS 服务 适用于 .NET 的 AWS SDK

以下各节包含示例、教程、任务和指南，向您展示如何使用 适用于 .NET 的 AWS SDK 来处理 AWS 服务。这些示例和教程依赖于 适用于 .NET 的 AWS SDK 提供的 API。要查看 API 中有哪些类和方法可用，请参阅适用于 .NET 的 AWS SDK API 参考。

如果您不熟悉 适用于 .NET 的 AWS SDK，则可能需要先查看该创建简单的应用程序主题。其中提供了对软件开发工具包的简单介绍。

您可以在代码示例存储库和 awslabs 存储库中找到更多AWS 代码示例。 GitHub

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

主题

- 的指导代码示例 适用于 .NET 的 AWS SDK
- AWS Lambda 用于计算服务
- 的高级库和框架 适用于 .NET 的 AWS SDK
- Support 对其他 AWS 服务和配置的支持

的指导代码示例 适用于 .NET 的 AWS SDK

以下各节包含代码示例，并提供示例指导。他们可以帮助您学习如何使用 适用于 .NET 的 AWS SDK 来处理 AWS 服务。有关其他代码示例，请参阅代码示例。

> **Note**
>
> 还提供了特定于 V3 适用于 .NET 的 AWS SDK 的代码示例。要找到它们，请参阅 适用于 .NET 的 AWS SDK (V3) 开发者指南中的代码示例和代码示例。但是，请注意，如果您在 SDK 的 V4（最新版本）中使用特定于 v3 的代码示例，则可能需要根据中的信息进行调整。迁移到版本 4

如果您不熟悉 适用于 .NET 的 AWS SDK，则可能需要先查看该创建简单的应用程序主题。其中提供了对软件开发工具包的简单介绍。

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

主题

- CloudFormation 使用访问 适用于 .NET 的 AWS SDK
- 使用 Amazon Cognito 验证用户身份
- 使用 Amazon DynamoDB NoSQL 数据库
- 使用 Amazon EC2
- 使用访问 AWS Identity and Access Management (IAM) 适用于 .NET 的 AWS SDK
- 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

CloudFormation 使用访问 适用于 .NET 的 AWS SDK

这些 适用于 .NET 的 AWS SDK 支持 AWS CloudFormation，可预测且重复地创建和配置 AWS 基础架构部署。

APIs

为 CloudFormation 客户 适用于 .NET 的 AWS SDK APIs 提供服务。 APIs 使您能够使用模板和堆栈等 CloudFormation 功能。本节包含少量示例，向您展示使用这些示例时可以遵循的模式 APIs。要查看全套内容 APIs，请参阅 适用于 .NET 的 AWS SDK API 参考（并滚动至 “Amazon. CloudFormation“）。

AWS CloudFormation APIs 它们由提供AWSSDK。 CloudFormation包裹。

先决条件

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

主题

主题

- 使用列出 AWS 资源 AWS CloudFormation

使用列出 AWS 资源 AWS CloudFormation

此示例向您展示如何使用列 适用于 .NET 的 AWS SDK 出 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 包裹：

- AWSSDK.CloudFormation

编程元素：

- 命名空间 Amazon。 CloudFormation

  班级 AmazonCloudFormationClient
- 命名空间 Amazon。 CloudFormation.Model

  班级ICloudFormationPaginatorFactory。 DescribeStacks

  班级 DescribeStackResourcesRequest

  班级 DescribeStackResourcesResponse

  类 Stack

  班级 StackResource

  类 Tag

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");
```

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

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

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

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

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

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
```

```
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

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

            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
```

```
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

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 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

使用 Amazon Cognito Identity，您可以为用户创建唯一身份，并对他们进行身份验证，以便安全访问您的 AWS 资源，例如亚马逊 S3 或 Amazon DynamoDB。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。该CognitoAuthentication 扩展库示例展示了如何使用 CognitoAuthentication 扩展库对 Amazon Cognito 用户池进行身份验证。

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 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

`Amazon.CognitoIdentity.CognitoAWSCredentials`，可在AWSSDK。 CognitoIdentity NuGetpack AWS age，是一个凭证对象，它使用 Amazon Cognito 和 AWS Security Token Service (AWS STS) 来检索用于拨打电话的证书。

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 Cognito AWSCredentials

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的身份调用 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();
}
```

AWS 以未经身份验证的用户身份使用

以下代码示例显示了如何以未经身份验证的用户身份开始使用 AWS ，然后通过 Facebook 进行身份验证并更新凭据以使用 Facebook 凭据。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 适用于 .NET 的 AWS SDK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 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

CognitoAuthentication 扩展程序库，可在 Amazon.Extensions 中找到。 CognitoAuthentication NuGet 包，简化了.NET Core 和 Xamarin 开发人员的 Amazon Cognito 用户池的身份验证过程。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

AmazonCognitoIdentityProviderClient使用 An onymou s 创建AWSCredentials，
它不需要签名请求。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

通过质询（例如使用 NewPasswordRequired 和多因素身份验证 (MFA)）来继续进行身份验证流程也更加简单。唯一的要求是 CognitoAuthentication 对象、用户的 SRP 密码以及下一个挑战的必要信息，这些信息是在提示用户输入后获取的。以下代码显示了一种在身份验证流程中检查质询类型并获得 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.");
    }

}
```

身份验证后使用 AWS 资源

使用 CognitoAuthentication 库对用户进行身份验证后，下一步就是允许该用户访问相应的 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，使用相应的身份池、用户池和 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 之外，扩展库还为 CognitoAuthentication 以下用户提供了更简单的身份验证流程：

- 自定义- 通过调用 `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)` 启动
- RefreshToken -首先致电至 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP-通过拨打以下电话启动 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP-通过拨打以下电话启动 `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

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。

适用于 .NET 的 AWS SDK 支持 Amazon DynamoDB，这是一项由提供的快速 NoSQL 数据库服务。AWS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模型 APIs，提供了如何以及何时使用它们的示例，并提供了指向中其他 DynamoDB 编程资源的链接。 适用于 .NET 的 AWS SDK

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.Dynamo DBv2 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. DBv2 Dynamo 访问此模型。 DocumentModel命名空间。

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 method

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 适用于 .NET 的 AWS SDK。

TableBuilder

AWSSDK. DBv2 NuGet Dynamo 软件包的 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 适用于 .NET 的 AWS SDK。

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。为了确定要获得的回复，该GetItemAsync方法使用目标回复 hash-and-range的主键。

```
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. DBv2 Dynamo 访问此模型。 DataModel命名空间。

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

在下面的类定义示例中，DynamoDBTable属性指定表名，而DynamoDBHashKey和DynamoDBRangeKey属性则建模表 hash-and-range的主键。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);
```

与 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，这些属性将通过对 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 适用于 .NET 的 AWS SDK。

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它信息

上面显示的示例和解释有时包括名为 DisableFetchingTableMetadata 的 DynamoDBContext 类的属性。此属性是在 AWSSDK.Dynamo DBv2 NuGet 包的 3.7.203 版本中引入的，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 适用于 .NET 的 AWS SDK。

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。该TableBuilder类也是在 AWSSDK.Dynamo 软件包的 3.7.203 版本中引入的。DBv2 NuGet

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

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

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();
```

```
// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
    IndexName = "PostedBy-Message-index"
});
```

更多信息

使用编程 DynamoDB、信息和示例 适用于 .NET 的 AWS SDK

- DynamoDB APIs
- 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
- 在 Amazon DynamoDB 上使用表达式和 适用于 .NET 的 AWS SDK
- Amazon DynamoDB 中的 JSON 支持

低级模型、信息和示例

- 使用 适用于 .NET 的 AWS SDK 低级 API 处理表
- 使用 适用于 .NET 的 AWS SDK 低级 API 处理项目
- 使用 适用于 .NET 的 AWS SDK 低级 API 查询表
- 使用 适用于 .NET 的 AWS SDK 低级 API 扫描表格
- 使用 适用于 .NET 的 AWS SDK 低级 API 处理本地二级索引
- 使用 适用于 .NET 的 AWS SDK 低级 API 使用全局二级索引

文档模型、信息和示例

- DynamoDB 数据类型

- 迪纳摩 DBEntry
- .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

其他有用信息

- 有关通过 .NET Aspire 使用 Amazon DynamoDB local 进行开发的信息，请参阅集成.NE AWS T Aspire。

主题

- 在 Amazon DynamoDB 上使用表达式和 适用于 .NET 的 AWS SDK
- Amazon DynamoDB 中的 JSON 支持

在 Amazon DynamoDB 上使用表达式和 适用于 .NET 的 AWS SDK

> **Note**
>
> 本主题中的信息特定于基于.NET Framework 和 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

以下代码示例演示如何使用通过表达式对 DynamoDB 适用于 .NET 的 AWS SDK 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。这些项目基于案例研究：A lte ProductCatalog m 中提供的示例。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.Write("\n");
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
```

```
  }

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

在前面的示例中，每个属性值都有多个 data-type-specific属性，可以通过评估这些属性来确定打印属性的正确格式。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。对于诸如BS、、L、MNSSS、和之类的属性，如果Count大于零，则该属性属于相应 non-zero-value的数据类型。如果该属性的所有属性均为null或Count等于零，则该属性对应于BOOL数据类型。 data-type-specific

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 适用于 .NET 的 AWS SDK 低级 API 处理项目
- 使用 适用于 .NET 的 AWS SDK 低级 API 查询表
- 使用 适用于 .NET 的 AWS SDK 低级 API 扫描表格
- 使用 适用于 .NET 的 AWS SDK 低级 API 处理本地二级索引
- 使用 适用于 .NET 的 AWS SDK 低级 API 使用全局二级索引

Amazon DynamoDB 中的 JSON 支持

> **Note**
>
> 本主题中的信息特定于基于.NET Framework 和 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

使用亚马逊 DynamoDB 时 适用于 .NET 的 AWS SDK 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.Write(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。为了确定要获取的项目，在本示例中，该GetItem方法使用目标项目 hash-and-range的主键。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 DynamoDB 编程 JSON 的更多信息和示例， 适用于 .NET 的 AWS SDK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

适用于 .NET 的 AWS SDK 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。

APIs

为 Amazon EC2 客户 适用于 .NET 的 AWS SDK 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，向您展示使用这些示例时可以遵循的模式 APIs。要查看全套内容 APIs，请参阅 适用于 .NET 的 AWS SDK API 参考（并滚动至“Amazon.ec2”）。

亚马逊 EC2 APIs 由 AWSSDK.EC2 NuGet 软件包提供。

先决条件

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK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 Amazon EC2 用户指南中的 Amazon EC2 安全组。

> **Warning**
>
> EC2-Classic 已于 2022 年 8 月 15 日停用。我们建议您从 EC2-Classic 迁移到 VPC。有关更多信息，请参阅博客文章 EC2-Classic Networking is Retiring - Here's How to Prepare。

有关 APIs 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

此示例向您展示如何使用枚举安全组。 适用于 .NET 的 AWS SDK 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($"\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();
```

```
    }
  }
```

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.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

  班级 DescribeSecurityGroupsRequest

  班级 DescribeSecurityGroupsResponse

  类 Filter

  班级 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。此名称来自对DescribeSecurityGroupsRequest类`Filters`属性的描述。

- 要获取安全组的完整列表，也可以不 DescribeSecurityGroupsAsync 带任何参数使用。

- 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

此示例向您展示如何使用创建安全组。 适用于 .NET 的 AWS SDK 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-Classic（前提是您的 AWS 账户支持）。

如果您未提供 VPC ID 且您的 AWS 账户不支持 EC2-Classic，则新的安全组将属于您账户的默认 VPC。

> **Warning**
>
> EC2-Classic 已于 2022 年 8 月 15 日停用。我们建议您从 EC2-Classic 迁移到 VPC。有关更多信息，请参阅博客文章 EC2-Classic Networking is Retiring – Here's How to Prepare。

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(
      $"\nOne or more security groups with name {groupName} already exist.\n");
    return securityGroups;
  }

  // If the security group doesn't already exists, create it.
  var createRequest = new CreateSecurityGroupRequest{
    GroupName = groupName
  };
  if(string.IsNullOrEmpty(vpcID))
  {
    createRequest.Description = "My .NET example security group for EC2-Classic";
  }
  else
  {
    createRequest.VpcId = vpcID;
    createRequest.Description = "My .NET example security group for EC2-VPC";
  }
  CreateSecurityGroupResponse createResponse =
    await ec2Client.CreateSecurityGroupAsync(createRequest);

  // Return the new security group
  DescribeSecurityGroupsResponse describeResponse =
    await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
      GroupIds = new List<string>() { createResponse.GroupId }
    });
```

```
        return describeResponse.SecurityGroups;
    }
```

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.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

  班级 CreateSecurityGroupRequest

  班级 CreateSecurityGroupResponse

  班级 DescribeSecurityGroupsRequest

  班级 DescribeSecurityGroupsResponse

  类 Filter

  班级 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($"\nNot a valid VPC ID: {vpcID}");

      // groupName has a value and vpcID either has a value or is null (which is fine)
      // Create the new security group and display information about it
      var securityGroups =
        await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
      Console.WriteLine("Information about the security group(s):");
      foreach(var group in securityGroups)
      {
        Console.WriteLine($"\nGroupName: {group.GroupName}");
        Console.WriteLine($"GroupId: {group.GroupId}");
        Console.WriteLine($"Description: {group.Description}");
        Console.WriteLine($"VpcId (if any): {group.VpcId}");
      }
    }
```

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

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
      }
      else
      {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }


    //
```

```
    // Method to determine if a security group with the specified name
    // already exists in the VPC
    private static async Task<List<SecurityGroup>> FindSecurityGroups(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      var request = new DescribeSecurityGroupsRequest();
      request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
      });
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n     If vpc-id isn't present, the security group will be" +
        "\n     for EC2-Classic (if your AWS account supports this)" +
        "\n     or will use the default VCP for EC2-VPC.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
```

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

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

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

     return parsedArgs;
   }

   //
```

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

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

}
```

更新安全组

此示例向您展示如何使用 适用于 .NET 的 AWS SDK 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($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
  Console.WriteLine($"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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

  班级 AuthorizeSecurityGroupIngressRequest

  班级 AuthorizeSecurityGroupIngressResponse

  班级 IpPermission

  班级 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($"\nThe given TCP port number, {portStr}, isn't
allowed.");

      // Add a rule to the given security group that allows
      // inbound traffic on a TCP port
      await AddIngressRule(
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
    }


    //
    // Method that adds a TCP ingress rule to a security group
    private static async Task AddIngressRule(
      IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
    {
      // Create an object to hold the request information for the rule.
      // It uses an IpPermission object to hold the IP information for the rule.
      var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
      ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
      });

      // Create the inbound rule for the security group
```

```
      AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
      Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }


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


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
```

```
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

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

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

      return parsedArgs;
    }

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

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

}
```

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”。的值`IpProtocol`可以在IpPermission类`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/。要了解有关授权入站流量的更多信息，请参阅 Amazon EC2 用户指南中的向安全组添加规则和针对不同使用案例的安全组规则。
  - 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。有关 Amazon EC2 密钥对的信息，请参阅 Amazon EC2 用户指南中的使用 Amazon EC2 密钥对。

有关 APIs 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

此示例向您展示如何使用创建密钥对。 适用于 .NET 的 AWS SDK 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($"\nCreated new key pair: {response.KeyPair.KeyName}");

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

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 (KeyPairInfo 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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端

- 命名空间 Amazon.EC2.Model

  班级 CreateKeyPairRequest

  班级 CreateKeyPairResponse

  班级 DescribeKeyPairsResponse

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

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


  //
```

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


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
 extension.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
 it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
```

```
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

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

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

      return parsedArgs;
    }

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

```
    }

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

}
```

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

此示例向您展示如何使用删除密钥对。 适用于 .NET 的 AWS SDK 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 (KeyPairInfo 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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

  班级 DeleteKeyPairRequest

班级 DescribeKeyPairsResponse

班级 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 (KeyPairInfo 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。

有关区域和可用区的更多信息，请参阅 Amazon EC2 用户指南中的区域和可用区。

此示例向您展示如何使用获取与 EC2 客户端相关的区域和可用区的详细信息。 适用于 .NET 的 AWS SDK 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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

Amazon EC2 客户端

- 命名空间 Amazon.EC2.Model

  班级 DescribeAvailabilityZonesResponse

  班级 DescribeRegionsResponse

  班级 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

您可以使用通过创建、启动和终止等操作 适用于 .NET 的 AWS SDK 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。要了解有关 EC2 实例的更多信息，请参阅亚马逊 EC2 用户指南中的亚马逊 EC2 实例。

有关 APIs 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

此示例向您展示如何使用从同一个亚马逊系统映像 (AMI) 启动一个或多个配置相同的 Amazon EC2 实例。 适用于 .NET 的 AWS SDK 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。

> **Warning**
>
> EC2-Classic 已于 2022 年 8 月 15 日停用。我们建议您从 EC2-Classic 迁移到 VPC。有关更多信息，请参阅博客文章 EC2-Classic Networking is Retiring – Here's How to Prepare。

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。要查找或创建它，一种简单的方法是登录 Amazon VPC 控制台，但您也可以使用CreateSubnetAsync和DescribeSubnetsAsync方法以编程方式获取它。

  > **Note**
  >
  > 如果您没有提供此参数，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。有关信息 AMIs，请参阅 Amazon EC2 用户指南中的亚马逊系统映像 (AMIs)。具体而言，请参阅查找 AMI 并共享 AMIs。

- 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。

有关该InstanceState属性的有效值，Instance.State.Code请参阅类。

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

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

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

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.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端

  班级 InstanceType
- 命名空间 Amazon.EC2.Model

  班级 DescribeInstancesRequest

  班级 DescribeInstancesResponse

  类 Instance

  班级 InstanceNetworkInterfaceSpecification

  班级 RunInstancesRequest

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

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

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


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
```

```
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


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

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

```
          parsedArgs.Add(key, value);
        }

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

      return parsedArgs;
    }

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

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

```
}
```

其他注意事项

- 检查 EC2 实例的状态时，您可以向DescribeInstancesRequest对象的`Filter`属性添加筛选器。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。

- 可用于RunInstancesRequest对象`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，有几个实例类型可供选择。有关更多信息，请参阅 Amazon EC2 用户指南中的 Amazon EC2 实例类型。另请参阅实例类型详细信息和实例类型浏览器。

- 您还可以在启动实例时将 IAM 角色附加到实例。为此，请创建一个其`Name`属性设置为 IAM 角色名称的IamInstanceProfileSpecification对象。然后将该对象添加到该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 Amazon EC2 用户指南中的授予用户将 IAM 角色传递给实例的权限。

（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《Amazon EC2 用户指南》中的连接到 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《Amazon EC2 用户指南》中的连接到 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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

  班级 GetPasswordDataRequest

  班级 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($"\nPassword: {password}");
    }


    //
    // Method to get the administrator password of a Windows EC2 instance
    private static async Task<string> GetPassword(
      IAmazonEC2 ec2Client, string instanceID, string pemFilename)
    {
```

```
      string password = string.Empty;
      GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
          InstanceId = instanceID});
      if(response.PasswordData != null)
      {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
      }
      else
      {
        Console.WriteLine($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
      }
      return password;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\n  -i, --instance-id: The name of the EC2 instance." +
        "\n  -p, --pem-filename: The name of the PEM file with the private key.");
    }
  }

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

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

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

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

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
```

```
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

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

}
```

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。

此示例向您展示如何使用终止 EC2 实例。 适用于 .NET 的 AWS SDK 它以实例 ID 作为输入。

SDK 参考

NuGet 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端
- 命名空间 Amazon.EC2.Model

班级 TerminateInstancesRequest

班级 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

本教程向您展示如何使用管理 Amazon EC2 竞价型实例。 适用于 .NET 的 AWS SDK

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。要更好地了解竞价型实例，请参阅 Amazon EC2 用户指南中的竞价型实例。

关于本教程

在学习本教程时，您可以使用 适用于 .NET 的 AWS SDK 来执行以下操作：

- 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 APIs 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。有几个实例类型可供选择。有关更多信息，请参阅《Amazon EC2 用户指南》https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/中的 Amazon EC2 实例类型。另请参阅实例类型详细信息和实例类型浏览器。

  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。有关信息 AMIs，请参阅 Amazon EC2 用户指南中的亚马逊系统映像 (AMIs)。具体而言，请参阅查找 AMI 并共享 AMIs。

- 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。

以下解释参考了 Amazon EC2 用户指南中的竞价型实例定价历史记录。

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状态代码的更多信息，请参阅 Amazon EC2 用户指南中的竞价型实例请求状态。

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 包裹：

- AWSSDK.EC2

编程元素：

- 命名空间 Amazon.EC2

  Amazon EC2 客户端

  班级 InstanceType

- 命名空间 Amazon.EC2.Model

  班级 CancelSpotInstanceRequestsRequest

  班级 DescribeSpotInstanceRequestsRequest

  班级 DescribeSpotInstanceRequestsResponse

  班级 InstanceStateChange

  班级 LaunchSpecification

  班级 RequestSpotInstancesRequest

  班级 RequestSpotInstancesResponse

  班级 SpotInstanceRequest

  班级 TerminateInstancesRequest

  班级 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($"\nSpot Instance request {requestId} " +
            $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
          break;
        }
```

```
        // Wait a bit and try again, longer each time (1, 2, 4, ...)
        Thread.Sleep(wait);
        wait = wait * 2;
      }

      // Show the user how long it took to fulfill the Spot Instance request.
      TimeSpan span = DateTime.Now.Subtract(start);
      Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

      // Perform actions here as needed.
      // For this example, simply wait for the user to hit a key.
      // That gives them a chance to look at the EC2 console to see
      // the running instance if they want to.
      Console.WriteLine("Press any key to start the cleanup...");
      Console.ReadKey(true);

      // Cancel the request.
      // Do this first to make sure that the request can't be re-fulfilled
      // once the Spot Instance has been terminated.
      Console.WriteLine("Canceling Spot Instance request...");
      await CancelSpotInstanceRequest(ec2Client, requestId);

      // Terminate the Spot Instance that's running.
      Console.WriteLine("Terminating the running Spot Instance...");
      await TerminateSpotInstance(ec2Client, requestId);

      Console.WriteLine("Done. Press any key to exit...");
      Console.ReadKey(true);
    }



    //
    // Method to create a Spot Instance request
    private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
      IAmazonEC2 ec2Client, string amiId, string securityGroupName,
      InstanceType instanceType, string spotPrice, int instanceCount)
    {
      var launchSpecification = new LaunchSpecification{
        ImageId = amiId,
        InstanceType = instanceType
      };
      launchSpecification.SecurityGroups.Add(securityGroupName);
      var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
```

```
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
      };

      RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
      return result.SpotInstanceRequests[0];
    }


    //
    // Method to get information about a Spot Instance request, including the status,
    // instance ID, etc.
    // It gets the information for a specific request (as opposed to all requests).
    private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
      return describeResponse.SpotInstanceRequests[0];
    }


    //
    // Method to cancel a Spot Instance request
    private static async Task CancelSpotInstanceRequest(
      IAmazonEC2 ec2Client, string requestId)
    {
      var cancelRequest = new CancelSpotInstanceRequestsRequest();
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }


    //
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
```

```
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
  SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
    }
  }
}
```

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。

使用访问 AWS Identity and Access Management (IAM) 适用于 .NET 的 AWS SDK

适用于 .NET 的 AWS SDK 支持 AWS Identity and Access Management，这是一项 Web 服务，可让 AWS 客户在中管理用户和用户权限 AWS。

AWS Identity and Access Management (IAM) 用户是您在中创建的实体 AWS。实体代表与 AWS之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者的联合身份验证，例如 AWS IAM Identity Center。

APIs

适用于 .NET 的 AWS SDK 提供给 I APIs AM 客户端。 APIs 使您可以使用 IAM 功能，例如用户、角色和访问密钥。

本节包含少量示例，向您展示使用这些示例时可以遵循的模式 APIs。要查看全套内容 APIs，请参阅适用于 .NET 的 AWS SDK 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 APIs 由提供AWSSDK。 IdentityManagement NuGet包裹。

先决条件

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK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根据给定的 JSON 策略文档创建 IAM 托管策略。 适用于 .NET 的 AWS SDK 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 包裹：

- AWSSDK.IdentityManagement

编程元素：

- 命名空间 Amazon。 IdentityManagement

  班级 AmazonIdentityManagementServiceClient
- 命名空间 Amazon。 IdentityManagement.Model

  班级 CreatePolicyRequest

  班级 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($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  Arn: {response.Policy.Arn}");
    }


    //
    // Method to create an IAM policy from a JSON file
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
 jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
 document.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
```

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

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

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

```
      return parsedArgs;
    }

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

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

}
```

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：

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Id"  : "DotnetTutorialPolicy",
    "Statement" : [
      {
  ```

```
      "Sid" : "DotnetTutorialPolicyS3",
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "DotnetTutorialPolicyPolly",
      "Effect": "Allow",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": "*"
    }
  ]
}
```

- 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

此示例向您展示如何使用 适用于 .NET 的 AWS SDK 来显示策略文档。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 包裹：

- AWSSDK.IdentityManagement

编程元素：

- 命名空间 Amazon。 IdentityManagement

  班级 AmazonIdentityManagementServiceClient
- 命名空间 Amazon。 IdentityManagement.Model

  班级 GetPolicyVersionRequest

  班级 GetPolicyVersionResponse

  班级 ListPolicyVersionsRequest

  班级 ListPolicyVersionsResponse

  班级 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

本教程向您展示如何使用在 Amazon EC2 实例上启用 IAM 角色。 适用于 .NET 的 AWS SDK

概述

对的所有请求都 AWS 必须使用颁发的凭证进行加密签名。 AWS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。要详细了解使用 IAM 角色的优势，请参阅 Amazon EC2 用户指南中的 Amazon EC2 的 IAM 角色。另外，有关 IAM 角色的信息，请参阅 IAM 用户指南。

对于使用构建的应用程序 适用于 .NET 的 AWS SDK，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证书。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。这些凭据用于 AWS 从客户端对象对进行调用。

关于本教程

在学习本教程时，您可以使用 适用于 .NET 的 AWS SDK （和其他工具）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
- 创建一个 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。
- AWS 开发计算机上允许您访问存储桶的凭据。

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 证书的信息，请参阅使用进行身份验证 适用于 .NET 的 AWS SDK AWS。

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 包裹：

- AWSSDK.S3

编程元素：

- 命名空间 Amazon.S3

  类 AmazonS3Client
- 命名空间 Amazon.S3.Model

  班级 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。（但请务必在完成后恢复凭证。）

创建一个 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 AmazonS3 ReadOnlyAccess。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

  要使用 EC2 控制台启动实例，请参阅 Amazon EC2 用户指南中的使用新的启动实例向导启动实例。

  浏览启动页面时，您至少应展开高级详细信息窗格，以便指定之前在 IAM 实例配置文件中创建的 IAM 角色。

- 使用 适用于 .NET 的 AWS SDK

  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 Amazon EC2 用户指南中的授予用户将 IAM 角色传递给实例的权限。

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。

有关如何连接到实例的信息，请参阅 Amazon EC2 用户指南中的连接到 Linux 实例或连接到 Windows 实例。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到实例的信息，请参阅 Amazon EC2 用户指南中的连接到 Linux 实例（请参阅相应的子部分）或将文件传输到 Windows 实例。
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

适用于 .NET 的 AWS SDK 支持用于互联网存储的 Amazon S3。该服务旨在降低开发人员进行网络规模级计算的难度。

APIs

针 适用于 .NET 的 AWS SDK 对 Amazon S3 客户端。 APIs APIs 使您能够使用 Amazon S3 资源，例如存储桶和项目。要查看 Amazon S3 APIs 的完整套件，请参阅以下内容：

- 适用于 .NET 的 AWS SDK API 参考（然后滚动至 “Amazon.S3”）。
- Amazon.Extensions.S3.Encryption 文档

Amazon S3 APIs 由以下 NuGet 软件包提供：

- AWSSDK.S3
- Amazon.Extensions.S3.Encryption

先决条件

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

本文档中的示例

本文档中的以下主题向您展示了如何使用与 Amazon S3 配合使用。 适用于 .NET 的 AWS SDK

- 使用 KMS 密钥进行 S3 加密

其它文档中的示例

以下指向 Amazon S3 开发者指南的链接提供了更多示例，说明如何使用与 Amazon S3 配合使用。 适用于 .NET 的 AWS SDK

> **Note**
>
> 尽管这些示例和其他编程注意事项是为 适用于 .NET 的 AWS SDK 使用.NET Framework 的版本 3 创建的，但它们也适用于 适用于 .NET 的 AWS SDK 使用.NET Core 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 ACLs

- 创建存储桶
- 上传对象
- 使用@@ 高级别 API (Amazon.S3.Transfer) 进行分段上传。 TransferUtility)
- 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 适用于 .NET 的 AWS SDK 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

在中使用 AWS KMS 密钥进行 Amazon S3 加密 适用于 .NET 的 AWS SDK

此示例向您展示如何使用 AWS Key Management Service 密钥加密 Amazon S3 对象。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 A mazonS3 EncryptionClient V2 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（V1 到 V2）。

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 包裹：

- Amazon.Extensions.S3.Encryption

编程元素：

- 命名空间 Amazon.Extensions.S3.Encryption

  AmazonS3 EncryptionClient 级 V2

  AmazonS3 CryptoConfiguration 级 V2

  班级 CryptoStorageMode

EncryptionMaterialsV 2 级

- 命名空间 Amazon.Extensions.S3.Encryption.Primitives

  班级 KmsType

- 命名空间 Amazon.S3.Model

  班级 GetObjectRequest

  班级 GetObjectResponse

  班级 PutObjectRequest

- 命名空间 Amazon。 KeyManagementService

  班级 AmazonKeyManagementServiceClient

- 命名空间 Amazon。 KeyManagementService.Model

  班级 CreateKeyRequest

  班级 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($"\nThe given file {fileName} doesn't exist.");
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

      // Create a customer master key (CMK) and store the result
      CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
      var kmsEncryptionContext = new Dictionary<string, string>();
      var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

      // Create the object in the bucket, then display the content of the object
      var putObjectResponse =
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
      Stream stream = putObjectResponse.ResponseStream;
      StreamReader reader = new StreamReader(stream);
      Console.WriteLine(reader.ReadToEnd());
    }
```

```
    //
    // Method to create and encrypt an object in an S3 bucket
    static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
      EncryptionMaterialsV2 materials, string bucketName,
      string fileName, string itemName)
    {
      // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
      var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

      // Create, encrypt, and put the object
      await s3EncClient.PutObjectAsync(new PutObjectRequest
      {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
      {
        BucketName = bucketName,
        Key = itemName
      });
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
    }
```

```
  }

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

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

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

```
            n++;
          }
        }

        return parsedArgs;
      }

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

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

}
```

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。

- A mazonS3 EncryptionClient V2 类实现的接口与标准AmazonS3Client类相同。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 密钥作为主密钥的一个好处是，您无需存储和管理自己的主密钥；这是通过完成的 AWS。第二个优点是，的AmazonS3EncryptionClientV2 适用于 .NET 的 AWS SDK 类可以与的AmazonS3EncryptionClientV2类互操作。 适用于 Java 的 AWS SDK这意味着您可以使用加密 适用于 Java 的 AWS SDK 并使用解密 适用于 .NET 的 AWS SDK，反之亦然。

  > **Note**
  >
  > 的AmazonS3EncryptionClientV2类仅在元数据模式下运行时 适用于 .NET 的 AWS SDK 支持 KMS 主密钥。的AmazonS3EncryptionClientV2类的指令文件模式与的AmazonS3EncryptionClientV2类 适用于 .NET 的 AWS SDK 不兼容 适用于 Java 的 AWS SDK。

- 有关该AmazonS3EncryptionClientV2类的客户端加密以及信封加密的工作原理的更多信息，请参阅使用适用于 .NET 的 AWS SDK 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 3.3 及更早 适用于 .NET 的 AWS SDK 版本的项目。

适用于 .NET 的 AWS SDK 支持亚马逊简单通知服务 (Amazon SNS) Simple Notification Service，这是一项网络服务，使应用程序、最终用户和设备能够立即从云端发送通知。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。此示例使用默认值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);
            }
        }
    }
}
```

请参阅完整的示例，包括有关如何从命令行构建和运行示例的信息 GitHub。

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);
            }
        }
    }
}
```

请参阅完整的示例，包括有关如何从命令行构建和运行示例的信息 GitHub。

使用 Amazon SQS 发送消息

适用于 .NET 的 AWS SDK 支持亚马逊简单队列服务 (Amazon SQS) Simple Queue Service，这是一项消息队列服务，用于处理系统中组件之间的消息或工作流程。

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。

APIs

APIs 为亚马逊 SQS 客户 适用于 .NET 的 AWS SDK 提供服务。 APIs 使您可以使用 Amazon SQS 功能，例如队列和消息。本节包含少量示例，向您展示使用这些示例时可以遵循的模式 APIs。要查看全套内容 APIs，请参阅 适用于 .NET 的 AWS SDK API 参考（并滚动至 “Amazon.sqs”）。

亚马逊 SQS APIs 由 AWSSDK. NuGet SQS 软件包提供。

先决条件

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK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创建 Amazon SQS 队列。 适用于 .NET 的 AWS SDK 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 包裹：

- AWSSDK.SQS

编程元素：

- 命名空间 Amazon.SQS

  Amazon 上课 SQSClient

  班级 QueueAttributeName
- 命名空间 Amazon.SQS.Model

  班级 CreateQueueRequest

  班级 CreateQueueResponse

  班级 GetQueueAttributesResponse

  班级 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。
- 队列名称和队列区 URLs 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 CreateQueueRequest 适用于 .NET 的 AWS SDKAPI 参考或SetQueueAttributes《亚马逊简单队列服务 API 参考》。
- 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 Amazon SQSClient 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更新 Amazon SQS 队列。 适用于 .NET 的 AWS SDK 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 包裹：

- AWSSDK.SQS

编程元素：

- 命名空间 Amazon.SQS

  Amazon 上课 SQSClient

  班级 QueueAttributeName
- 命名空间 Amazon.SQS.Model

  班级 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($"\nThe given attribute name, {attribute}, isn't valid.");
        }
      }
    }
```

```
    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
          new List<string>{ QueueAttributeName.All });
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }


    //
    // Method to check the name of the attribute
    private static bool ValidAttribute(string attribute)
    {
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }


    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
```

```
      Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
 value]");
      Console.WriteLine("  -q: The URL of the queue you want to update.");
      Console.WriteLine("  -a: The name of the attribute to update.");
      Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
  }


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

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

```
        }

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

      return parsedArgs;
    }

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

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

}
```

其他注意事项

- 要更新该RedrivePolicy属性，您必须根据自己的操作系统将整个值加上引号并 key/value 对对的引号进行转义。

  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删除 Amazon SQS 队列。 适用于 .NET 的 AWS SDK 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 包裹：

- AWSSDK.SQS

编程元素：

- 命名空间 Amazon.SQS

  Amazon 上课 SQSClient
- 命名空间 Amazon.SQS.Model

  班级 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。 适用于 .NET 的 AWS SDK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 包裹：

- AWSSDK.SQS

编程元素：

- 命名空间 Amazon.SQS

  Amazon 上课 SQSClient
- 命名空间 Amazon.SQS.Model

  班级 PurgeQueueResponse

  班级 SendMessageBatchResponse

  班级 SendMessageResponse

  班级 SendMessageBatchRequestEntry

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


    //
    // Method to get input from the user
    // They can provide messages to put in the queue or exit the application
    private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
    {
      string response;
      while (true)
      {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
```

```
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
      }
    }


    //
    // Method to delete all messages from the queue
    private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
    }
  }
}
```

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接收 适用于 .NET 的 AWS SDK 来自 Amazon SQS 队列的消息，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 包裹：

- AWSSDK.SQS

编程元素：

- 命名空间 Amazon.SQS

  Amazon 上课 SQSClient
- 命名空间 Amazon.SQS.Model

  班级 ReceiveMessageRequest

  班级 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 Amazon SQSClient 类ChangeMessageVisibilityAsync的方法。

- 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。

AWS Lambda 用于计算服务

适用于 .NET 的 AWS SDK 支持 AWS Lambda，使您无需预置或管理服务器即可运行代码。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。

APIs

适用于 .NET 的 AWS SDK 规定 APIs 了 AWS Lambda。 APIs 使您能够使用函数、触发器和事件等 Lambda 功能。要查看完整的 APIs，请参阅 适用于 .NET 的 AWS SDK API 参考中的 Lambda。

Lambda APIs 由软件包提供。NuGet

先决条件

在开始之前，请确保已经设置好环境并配置了项目。还要查看使用 SDK中的信息。

其他信息

有关 AWS Lambda 通过.NET Aspire 进行开发的信息，请参阅集成.NE AWS T Aspire。

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：

- 有关 Lambda 注释 APIs 和属性的文档的GitHub 自述文件。
- 有关 Lambda 注释的博客文章。
- The `Amazon.Lambda.Annotations` NuGet 程序包。
- 照片资产管理项目开启 GitHub。具体而言，请参阅项目自述PamApiAnnotations文件中的文件夹和对 Lambda 注释的引用。

> **Note**
>
> 前面的示例特定于 V3 的 适用于 .NET 的 AWS SDK。如果您在 SDK 的 V4（最新版本）中使用示例，则可能需要根据中的迁移到版本 4信息进行调整。

的高级库和框架 适用于 .NET 的 AWS SDK

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 适用于 .NET 的 AWS SDK，则可能需要先查看该创建简单的应用程序主题。其中提供了对软件开发工具包的简单介绍。

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

主题

- AWS .NET 的消息处理框架
- 在中 AWS 与.NET Aspire 集成 适用于 .NET 的 AWS SDK

AWS .NET 的消息处理框架

.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。有关该框架如何简化开发的详细信息，请参阅博客文章 .NET AWS 消息处理框架简介（预览版）。第一部分特别提供了演示，展示了使用低级 API 调用和使用框架之间的区别。

消息处理框架支持以下活动和功能：

- 向 SQS 发送消息并将事件发布到 SNS 和. EventBridge

- 使用长时间运行的轮询器接收和处理来自 SQS 的消息，该轮询器通常用于后台服务。这包括在处理消息时管理可见性超时，以防止其他客户端处理该消息。
- 处理 AWS Lambda 函数中的消息。
- FIFO (first-in-first-out) SQS 队列和 SNS 主题。
- OpenTelemetry 用于记录。

有关这些活动和功能的详细信息，请参阅博客文章的 “功能” 部分以及下面列出的主题。

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

其他资源

- NuGet.org 上的AWS.Messaging软件包。
- API 参考资料。
- GitHub 存储库中的README文件位于 https://github.com/aws/aws-dotnet-messaging/
- 来自微软的@@ .NET 依赖注入。
- 来自微软的@@ .NET 通用主机。

主题

- 开始使用适用于.NET 的 AWS 消息处理框架
- 使用适用于.NET 的 AWS 消息处理框架发布消息
- 使用适用于.NET 的 AWS 消息处理框架使用消息
- 将 FIFO 与适用于.NET 的 AWS 消息处理框架配合使用
- .NET AWS 消息处理框架的日志记录和开放遥测
- 为.NET 自定义 AWS 消息处理框架
- .NET AWS 消息处理框架的安全性

开始使用适用于.NET 的 AWS 消息处理框架

在开始之前，请确保您已经设置了环境并配置了项目。还要查看使用 SDK中的信息。

本主题提供的信息将帮助您开始使用消息处理框架。除了先决条件和配置信息外，还提供了一个教程，向您展示如何实现常见场景。

先决条件和配置

- 您为应用程序提供的凭证必须对应用程序所使用的消息服务和操作具有相应的权限。有关更多信息，请参阅 SQS、SN S 的安全主题及其各自EventBridge的开发者指南。另请参阅 README 文件中讨论特定权限 GitHub 的部分。
- 要使用适用于.NET 的 AWS 消息处理框架，必须将该AWS.Messaging NuGet包添加到您的项目中。例如：

```
dotnet add package AWS.Messaging
```

- 该框架与.NET 的依赖注入 (DI) 服务容器集成。您可以在应用程序启动期间通过调用将其添加AddAWSMessageBus到 DI 容器来配置框架。

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

教程

本教程演示如何使用适用于.NET 的 AWS 消息处理框架。它创建了两个应用程序：一个 ASP.NET Core Minimal API，用于在 API 终端节点收到请求时向 Amazon SQS 队列发送消息，以及一个长时间运行的控制台应用程序，用于轮询和处理这些消息。

- 本教程中的说明偏向于.NET CLI，但你可以使用跨平台工具（例如.NET CLI 或 Microsoft Visual Studio）来执行本教程。有关工具的信息，请参见安装和配置您的工具链 适用于 .NET 的 AWS SDK。
- 本教程假设您正在使用[default]个人资料作为凭证。它还假设短期证书具有发送和接收 Amazon SQS 消息的相应权限。有关更多信息，请参阅使用进行身份验证 适用于 .NET 的 AWS SDK AWS和 S QS 的安全主题。

> **Note**
>
> 运行本教程后，您可能会产生 SQS 消息传送费用。

步骤

- 创建 SQS 队列
- 创建并运行发布应用程序
- 创建并运行处理应用程序
- 清理

创建 SQS 队列

本教程需要一个 SQS 队列来向其发送消息和从中接收消息。可以使用 AWS CLI 或的以下命令之一来创建队列 AWS Tools for PowerShell。记下返回的队列 URL，以便可以在随后的框架配置中指定它。

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

AWS Tools for PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

创建并运行发布应用程序

使用以下步骤创建和运行发布应用程序。

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 webapi --name Publisher
```

3. 导航到新项目的文件夹。添加对.NET AWS 消息处理框架的依赖关系。

```
cd Publisher
dotnet add package AWS.Messaging
```

> **Note**
>
> 如果您使用身份 AWS IAM Identity Center 验证，请务必同时添加`AWSSDK.SSO`和`AWSSDK.SSOOIDC`。

4. 将中的`Program.cs`代码替换为以下代码。

```
using AWS.Messaging;
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/
swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
    // The SQS URL should be passed as a command line argument or set in the Debug
 launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

```
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
 Publisher.GreetingMessage message) =>
    {
        return await PostGreeting(message, publisher);
    })
.WithName("SendGreeting")
.WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}

namespace Publisher
{
    /// <summary>
```

```
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }
}
```

5. 运行以下命令。这应该会打开一个带有 Swagger UI 的浏览器窗口，允许你浏览和测试你的 API。

```
dotnet watch run <queue URL created earlier>
```

6. 打开/greeting端点并选择试用。
7. 为消息指定senderName和greeting值，然后选择执行。这会调用你的 API，它会发送 SQS 消息。

创建并运行处理应用程序

使用以下过程创建和运行处理应用程序。

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 Handler
```

3. 导航到新项目的文件夹。添加对.NET AWS 消息处理框架的依赖关系。还要添加Microsoft.Extensions.Hosting软件包，它允许您通过 .NET 通用主机配置框架。

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> 如果您使用身份 AWS IAM Identity Center 验证，请务必同时添加AWSSDK.SSO和AWSSDK.SSOOIDC。

4. 将中的Program.cs代码替换为以下代码。

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
 Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);

            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
 GreetingMessage>("greetingMessage");

        }
        // You can add additional message handlers here, using different message
 types.
    });
});

var host = builder.Build();
await host.RunAsync();

namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
```

```
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from
 {messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. 运行以下命令。这将启动一个长期运行的民意调查员。

```
dotnet run <queue URL created earlier>
```

启动后不久，应用程序将收到本教程第一部分中发送的消息并记录以下消息：

```
Received message {greeting} from {senderName}
```

6. 按下Ctrl+C可停止轮询器。

清理

使用 AWS CLI 或的以下命令之一 AWS Tools for PowerShell 删除队列。

AWS CLI

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

AWS Tools for PowerShell

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

使用适用于.NET 的 AWS 消息处理框架发布消息

.NET AWS 消息处理框架支持发布一种或多种消息类型、处理一种或多种消息类型，或者在同一个应用程序中同时处理这两种消息类型。

以下代码显示了向不同 AWS 服务发布不同消息类型的应用程序的配置。

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
 EventBridge bus
    builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-
west-2:012345678910:event-bus/default");
});
```

在启动期间注册框架后，IMessagePublisher将泛型注入到代码中。调用其PublishAsync方法来发布上面配置的任何消息类型。通用发布者将根据消息的类型确定要将消息路由到的目的地。

在以下示例中，ASP.NET MVC 控制器接收来自用户的ChatMessage消息和OrderInfo事件，然后分别将其发布到亚马逊 SQS 和亚马逊 SNS。两种消息类型都可以使用上面配置的通用发布器发布。

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;
```

```
    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

为了将消息路由到相应的处理逻辑，框架使用称为消息类型标识符的元数据。默认情况下，这是消息的.NET 类型的全名，包括其程序集名称。如果你既要发送消息，又要处理消息，那么如果你跨项目共

享消息对象的定义，这种机制就能很好地发挥作用。但是，如果在不同的命名空间中重新定义消息，或者您要与其他框架或编程语言交换消息，则可能需要覆盖消息类型标识符。

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
        // Register that you'll publish messages of type GreetingMessage to an existing
 queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

特定服务的出版商

上面显示的示例使用通用IMessagePublisher，它可以根据配置的消息类型发布到任何支持的 AWS 服务。该框架还为亚马逊 SQS、Amazon SNS 和亚马逊提供特定服务的发布商。 EventBridge这些特定的发布者公开的选项仅适用于该服务，并且可以使用ISQSPublisherISNSPublisher、和类型进行注入IEventBridgePublisher。

例如，向 SQS FIFO 队列发送消息时，必须设置相应的消息组 ID。以下代码再次显示了该ChatMessage示例，但现在使用ISQSPublisher来设置特定于 SQS 的选项。

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
```

```
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-
specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
            MessageDeduplicationId = <message-deduplication-id>,
            MessageGroupId = <message-group-id>
        });

        return Ok();
    }
}
```

对于 SNS 和 EventBridge，也可以`IEventBridgePublisher`分别使用`ISNSPublisher`和。

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

默认情况下，给定类型的消息会发送到预先配置的目的地。但是，您可以使用特定于消息的发布者来覆盖单条消息的目的地。您还可以覆盖用于发布消息的底层 适用于 .NET 的 AWS SDK 客户端，这在需要根据目标更改角色或凭据的多租户应用程序中非常有用。

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

使用适用于.NET 的 AWS 消息处理框架使用消息

.NET AWS 消息处理框架允许您使用该框架或其中一个消息传递服务发布的消息。可以通过多种方式使用消息，其中一些方式如下所述。

消息处理器

要使用消息，请使用要处理的每种消息类型的`IMessageHandler`接口实现消息处理程序。消息类型和消息处理程序之间的映射是在项目启动时配置的。

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd");

            // Register all IMessageHandler implementations with the message type they
 should process.
            // Here messages that match our ChatMessage .NET type will be handled by
 our ChatMessageHandler
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

以下代码显示了消息的示例消息处理程序。ChatMessage

```
public class ChatMessageHandler : IMessageHandler<ChatMessage>
{
    public Task<MessageProcessStatus> HandleAsync(MessageEnvelope<ChatMessage>
 messageEnvelope, CancellationToken token = default)
    {
        // Add business and validation logic here.
        if (messageEnvelope == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        if (messageEnvelope.Message == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        ChatMessage message = messageEnvelope.Message;

        Console.WriteLine($"Message Description: {message.MessageDescription}");

        // Return success so the framework will delete the message from the queue.
        return Task.FromResult(MessageProcessStatus.Success());
    }
}
```

外部MessageEnvelope包含框架使用的元数据。它的message属性是消息类型（在本例中ChatMessage）。

您可以返回MessageProcessStatus.Success()以表明消息已成功处理，框架将从 Amazon SQS 队列中删除该消息。返回时MessageProcessStatus.Failed()，消息将保留在队列中，可以在该队列中再次处理或移至死信队列（如果已配置）。

在长时间运行的进程中处理消息

您可以使用 SQS 队列 URL 调AddSQSPoller用以启动一个长时间运行的队列[BackgroundService](#)，该队列将持续轮询队列并处理消息。

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
```

```
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
            {
                // The maximum number of messages from this queue that the framework
 will process concurrently on this client.
                options.MaxNumberOfConcurrentMessages = 10;

                // The duration each call to SQS will wait for new messages.
                options.WaitTimeSeconds = 20;
            });

            // Register all IMessageHandler implementations with the message type they
 should process.
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

配置 SQS 消息轮询器

调用SQSMessagePollerOptions时可以配置 SQS 消息轮询器。AddSQSPoller

- MaxNumberOfConcurrentMessages-队列中可同时处理的最大消息数。默认值是 10。
- WaitTimeSeconds-S ReceiveMessage QS 调用等待消息到达队列后再返回的持续时间（以秒为单位）。如果有消息可用，则该呼叫的返回时间早于WaitTimeSeconds。默认值为 20。

消息可见性超时处理

SQS 消息有可见性超时时间。当一个使用者开始处理给定消息时，它会保留在队列中，但为了避免多次处理该消息，其他消费者会将其隐藏。如果消息在再次可见之前未被处理和删除，则其他使用者可能会尝试处理同一条消息。

该框架将跟踪并尝试延长其当前正在处理的消息的可见性超时时间。您可以在 “呼叫SQSMessagePollerOptions时” 上配置此行为AddSQSPoller。

- `VisibilityTimeout`-在后续检索请求中隐藏收到消息的持续时间（以秒为单位）。默认值为 30。
- `VisibilityTimeoutExtensionThreshold`-当消息的可见性超时在这几秒钟内到期时，框架将延长可见性超时（再延长`VisibilityTimeout`几秒钟）。默认值是 5。
- `VisibilityTimeoutExtensionHeartbeatInterval`-框架检查即将到期的几秒钟之内的消息，然后延长其可见性超时的频率（以`VisibilityTimeoutExtensionThreshold`秒为单位）。默认值是 1。

在以下示例中，框架将每 1 秒检查一次仍在处理的消息。对于这些消息在再次可见后 5 秒钟内，框架会自动将每条消息的可见性超时时间再延长 30 秒。

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
 options =>
{
    options.VisibilityTimeout = 30;
    options.VisibilityTimeoutExtensionThreshold = 5;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

处理 AWS Lambda 函数中的消息

您可以将适用于.NET 的 AWS 消息处理框架与 SQS 与 Lambda 集成。这是由`AWS.Messaging.Lambda`软件包提供的。请参考其自述文件开始使用。

将 FIFO 与适用于.NET 的 AWS 消息处理框架配合使用

对于消息排序和消息重复数据删除至关重要的用例，.NET AWS 消息处理框架支持 first-in-first-out (FIFO) Amazon SQS 队列和 Amazon SNS 主题。

发布

将消息发布到 FIFO 队列或主题时，必须设置消息组 ID，该标识指定消息所属的组。群组内的消息按顺序处理。您可以在特定于 SQS 和 SNS 特定的消息发布器上进行此设置。

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

订阅

处理来自 FIFO 队列的消息时，框架会按照每次ReceiveMessages调用的接收顺序处理给定消息组中的消息。当配置了以结尾的队列时，框架会自动进入此操作模式.fifo。

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET.
        services.AddAWSMessageBus(builder =>
        {
            // Because this is a FIFO queue, the framework automatically handles these
 messages in order.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MPF.fifo");
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();
        });
    })
    .Build()
    .RunAsync();
```

.NET AWS 消息处理框架的日志记录和开放遥测

.NET 的 AWS 消息处理框架 OpenTelemetry 用于记录该框架发布或处理的每条消息的跟踪。这是由AWS.Messaging.Telemetry.OpenTelemetry软件包提供的。请参考其自述文件开始使用。

> **Note**
>
> 有关日志记录的安全信息，请参阅.NET AWS 消息处理框架的安全性。

为.NET 自定义 AWS 消息处理框架

.NET 的 AWS 消息处理框架在三个不同的 “层” 中构建、发送和处理消息：

1. 在最外层，框架构建特定于服务的 AWS-native 请求或响应。例如，在 Amazon SQS 中，它可以生成SendMessage请求并处理服务定义的Message对象。
2. 在 SQS 请求和响应中，框架将MessageBody元素（或者Message对于亚马逊 SNS 或亚马逊）设置Detail为 JS EventBridge ON 格式。 CloudEvent它包含框架设置的元数据，处理消息时可以在MessageEnvelope对象上访问这些元数据。

3. 在最内层， CloudEvent JSON 对象内的data属性包含作为消息发送或接收的.NET 对象的 JSON 序列化。

```
{
    "id":"b02f156b-0f02-48cf-ae54-4fbbe05cffba",
    "source":"/aws/messaging",
    "specversion":"1.0",
    "type":"Publisher.Models.ChatMessage",
    "time":"2023-11-21T16:36:02.8957126+00:00",
    "data":"<the ChatMessage object serialized as JSON>"
}
```

您可以自定义邮件信封的配置方式和读取方式：

- "id"唯一标识消息。默认情况下，它设置为新的 GUID，但是可以通过实现自己的GUID IMessageIdGenerator 并将其注入到DI容器中来覆盖它。
- "type"控制如何将消息路由到处理程序。默认情况下，它使用与消息对应的.NET 类型的全名。当通过AddSQSPublisher、或将消息类型映射到目标时AddSNSPublisher，您可以通过messageTypeIdentifier参数覆盖此设置AddEventBridgePublisher。
- "source"表示哪个系统或服务器发送了消息。
  - 如果从中发布，这将是函数名称；如果在 Amazon ECS 上发布 AWS Lambda，则为集群名称和任务 ARN；如果在 Amazon 上，则为实例 ID EC2，否则为后备值。/aws/messaging
  - 您可以通过AddMessageSource或AddMessageSourceSuffix在上覆盖此设置MessageBusBuilder。
- "time"在 UTC DateTime 中设置为当前。这可以通过实现自己的内容并将其注入到 DI IDateTimeHandler 容器中来覆盖。
- "data"包含作为消息发送或接收的.NET 对象的 JSON 表示形式：
  - ConfigureSerializationOptionson MessageBusBuilder 允许您配置序列化和反序列化消息时将使用的。System.Text.Json.JsonSerializerOptions
  - 要注入其他属性或在框架构建消息封套后对其进行转换，您可以通过 AddSerializationCallback on 实现ISerializationCallback和注册该属性MessageBusBuilder。

.NET AWS 消息处理框架的安全性

.NET 的 AWS 消息处理框架依赖于与 适用于 .NET 的 AWS SDK 进行通信 AWS。有关中安全性的更多信息 适用于 .NET 的 AWS SDK，请参阅为此提供安全保障 AWS 产品或服务。

出于安全考虑，该框架不记录用户发送的数据消息。如果要出于调试目的启用此功能，则需要按如下方式调用 `EnableDataMessageLogging()` Message Bus：

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

如果您发现潜在的安全问题，请参阅安全策略以获取报告信息。

在中 AWS 与.NET Aspire 集成 适用于 .NET 的 AWS SDK

.NET Aspire 是一种构建云就绪应用程序的新方法。特别是，它为本地环境提供了一种编排，可以在其中运行、连接和调试分布式应用程序的组件。为了改善云端应用程序的内部开发循环，我们创建了与.NET Aspire 的集成，用于将.NET 应用程序连接到 AWS 资源。这些集成可通过 aspire.hosting.aws 软件包 NuGet 获得。

以下.NET Aspire 集成可用：

- 能够通过配置您的 AWS 资源CloudFormation。这种集成是在.NET Aspire AppHost 项目中使用的。

  有关更多信息，请参阅博客文章AWS 与.NET Aspire 集成。
- 在本地安装、配置和连接亚马逊 DynamoDB。 适用于 .NET 的 AWS SDK 这种集成是在.NET Aspire AppHost 项目中使用的。

  有关更多信息，请参阅博客文章AWS 与.NET Aspire 集成。
- 为AWS Lambda函数启用本地开发环境。这种集成是在.NET Aspire AppHost 项目中使用的。

  有关更多信息，请参阅博客文章使用.NET Aspire 构建和调试.NET Lambda 应用程序（第 1 部分）和使用.NET Aspire 构建和调试.NET Lambda 应用程序（第 2 部分）。

> **Note**
>
> 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。

由于此功能处于预览阶段，因此您需要选择使用预览功能。有关此预览功能以及如何选择加入的更多信息，请参阅上的 GitHub开发跟踪器问题。

其他信息

有关如何使用A spire.hosting.aws 中提供的集成的更多信息和详细信息，请参阅以下资源。

- 博客文章AWS 与.NET Aspire 集成。
- 博客文章使用.NET Aspire 构建和调试.NET Lambda 应用程序（第 1 部分）以及使用.NET Aspire 构建和调试.NET Lambda 应用程序（第 2 部分）。
- integrations-on-dotnet-aspire-for-aws 存储库已启用。 GitHub
- aspi NuGet re.hosting.aws 软件包的详细自述文件。

Support 对其他 AWS 服务和配置的支持

除前面各节中描述的 AWS 服务外，还提供其他 适用于 .NET 的 AWS SDK 支持服务。有关所有支持的服务的信息，请参阅 适用于 .NET 的 AWS SDK API 参考。 APIs

除了单个 AWS 服务的命名空间外， 适用于 .NET 的 AWS SDK 还提供以下内容： APIs

区域图	描述	资源
AWS Support	以编程方式访问 AWS 支持案例和 Trusted Advisor 功能。	参见 Amazon。 AWSSupport还有亚马逊。 AWSSupport.Model。
常规	帮助程序类和枚举。	请参阅 Amazon 和 Amazon.Util。

适用于 .NET 的 SDK (v4) 代码示例

本主题中的代码示例向您展示了如何将 适用于 .NET 的 AWS SDK (v4) 与 AWS。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

某些服务包含其他示例类别，这些类别说明如何利用特定于服务的库或函数。

服务

- 使用 Aurora 的示例 适用于 .NET 的 SDK (v4)
- 使用的 Auto Scaling 示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon Bedrock 示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon Bedrock 运行时示例 适用于 .NET 的 SDK (v4)
- CloudFormation 使用示例 适用于 .NET 的 SDK (v4)
- CloudWatch 使用示例 适用于 .NET 的 SDK (v4)
- CloudWatch 使用记录示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon Cognito 身份提供商示例 适用于 .NET 的 SDK (v4)
- AWS Control Tower 使用示例 适用于 .NET 的 SDK (v4)
- 使用 DynamoDB 示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon EC2 的示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon ECS 示例 适用于 .NET 的 SDK (v4)
- AWS IoT 使用示例 适用于 .NET 的 SDK (v4)
- AWS IoT data 使用示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon Redshift 示例 适用于 .NET 的 SDK (v4)
- 使用 Amazon S3 的示例 适用于 .NET 的 SDK (v4)

使用 Aurora 的示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Aurora 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Aurora

以下代码示例显示如何开始使用 Aurora。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 DescribeDBClusters。

基本功能

了解基本功能

以下代码示例展示了如何：

- 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。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }
```

```
        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
```

```
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
```

```
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
 a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
```

```
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {
            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
                newInstance = instances.First();
            }
        }
```

```
        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
```

```
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
```

```
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

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

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
```

```
                        ynResponse.Equals("y",
                            StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

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.GetValueOrDefault() && p.DataType == "integer")
            {
```

```
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
```

```
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
```

```
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
 dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
```

```
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
```

```
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();
```

```
        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="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《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

操作

`CreateDBCluster`

以下代码示例演示了如何使用 `CreateDBCluster`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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《适用于 .NET 的 AWS SDK API Reference》中的 CreateDBCluster。

`CreateDBClusterParameterGroup`

以下代码示例演示了如何使用 `CreateDBClusterParameterGroup`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考[CreateDBClusterParameterGroup]中的。

`CreateDBClusterSnapshot`

以下代码示例演示了如何使用 `CreateDBClusterSnapshot`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考[CreateDBClusterSnapshot]中的。

`CreateDBInstance`

以下代码示例演示了如何使用 `CreateDBInstance`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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《适用于 .NET 的 AWS SDK API Reference》中的 CreateDBInstance。

DeleteDBCluster

以下代码示例演示了如何使用 DeleteDBCluster。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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《适用于 .NET 的 AWS SDK API Reference》中的 DeleteDBCluster。

DeleteDBClusterParameterGroup

以下代码示例演示了如何使用 DeleteDBClusterParameterGroup。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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> DeleteClusterParameterGroupByNameAsync(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 适用于 .NET 的 AWS SDK API 参考DeleteDBClusterParameterGroup中的。

DeleteDBInstance

以下代码示例演示了如何使用 DeleteDBInstance。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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《适用于 .NET 的 AWS SDK API Reference》中的 DeleteDBInstance。

`DescribeDBClusterParameterGroups`

以下代码示例演示了如何使用 `DescribeDBClusterParameterGroups`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeDBClusterParameterGroups中的。

`DescribeDBClusterParameters`

以下代码示例演示了如何使用 `DescribeDBClusterParameters`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考[DescribeDBClusterParameters](#)中的。

DescribeDBClusterSnapshots

以下代码示例演示了如何使用 DescribeDBClusterSnapshots。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeDBClusterSnapshots中的。

DescribeDBClusters

以下代码示例演示了如何使用 DescribeDBClusters。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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);
            if (response.DBClusters != null)
            {
                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《适用于 .NET 的 AWS SDK API Reference》中的 DescribeDBClusters。

`DescribeDBEngineVersions`

以下代码示例演示了如何使用 `DescribeDBEngineVersions`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeDBEngineVersions中的。

`DescribeDBInstances`

以下代码示例演示了如何使用 `DescribeDBInstances`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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《适用于 .NET 的 AWS SDK API Reference》中的 DescribeDBInstances。

`DescribeOrderableDBInstanceOptions`

以下代码示例演示了如何使用 `DescribeOrderableDBInstanceOptions`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeOrderableDBInstanceOptions中的。

`ModifyDBClusterParameterGroup`

以下代码示例演示了如何使用 `ModifyDBClusterParameterGroup`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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.GetValueOrDefault() && p.DataType == "integer")
           {
               while (newValue == 0)
               {
                   Console.WriteLine(
                       $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                   var choice = Console.ReadLine();
                   int.TryParse(choice, out newValue);
               }

               p.ParameterValue = newValue.ToString();
           }
       }
```

```
        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[ModifyDBClusterParameterGroup](#)中的。

使用的 Auto Scaling 示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Auto Scaling 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Auto Scaling

以下代码示例显示了如何开始使用 Auto Scaling。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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();

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

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

```
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[DescribeAutoScalingGroups](#)中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 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。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)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[0].ZoneName);

        // Keep checking the details of the new group until its lifecycle state
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
```

```
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);

        Console.WriteLine($"Auto scaling group {groupName} successfully created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");
```

```
        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are {activity.Details}");
            });
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
```

```
        if (metrics.Any())
        {
            Console.WriteLine($"Metrics collected for {groupName}:");
            metrics.ForEach(metric =>
            {
                Console.Write($"Metric name: {metric.MetricName}\t");
                Console.WriteLine($"Namespace: {metric.Namespace}");
            });
        }

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        if (dataPoints.Any())
        {
            Console.WriteLine("Details for the metrics collected:");
            dataPoints.ForEach(detail => { Console.WriteLine(detail); });
        }

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for {groupName}.");
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
        Console.WriteLine("Now terminating all instances in the Auto Scaling
group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
```

```
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
 autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
 ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
```

```
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
 the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
 group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;

        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
```

```
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);
```

```
        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

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

```
    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
 response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
 response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

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

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }
```

```
        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groups = new List<AutoScalingGroup>();
        var groupList = new List<string>
            {
                groupName,
            };

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

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        if (response.AutoScalingGroups != null)
        {
            groups = response.AutoScalingGroups;
        }

        return groups;
    }



    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
```

```
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

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

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


    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
{desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }


    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
```

```
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;

using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
```

```
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }

    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
```

```
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }

        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
```

```
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
    public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
    {
        var metrics = new List<Metric>();
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };
```

```
        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);
        if (response.Metrics != null)
        {
            metrics = response.Metrics;
        }
        return metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var dataPoints = new List<Datapoint>();
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
```

```
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
        if (response.Datapoints != null)
        {
            dataPoints = response.Datapoints;
        }

        return dataPoints;
    }

}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

操作

`CreateAutoScalingGroup`

以下代码示例演示了如何使用 `CreateAutoScalingGroup`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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
    };
    try
    {
```

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

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[CreateAutoScalingGroup](#)中的。

`DescribeAutoScalingGroups`

以下代码示例演示了如何使用 `DescribeAutoScalingGroups`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)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>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

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

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[DescribeAutoScalingGroups](#)中的。

DescribeAutoScalingInstances

以下代码示例演示了如何使用 DescribeAutoScalingInstances。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

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

```
            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[DescribeAutoScalingInstances](#)中的。

`DescribeScalingActivities`

以下代码示例演示了如何使用 `DescribeScalingActivities`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
```

```
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
_amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeScalingActivities中的。

`DisableMetricsCollection`

以下代码示例演示了如何使用 `DisableMetricsCollection`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DisableMetricsCollection中的。

`EnableMetricsCollection`

以下代码示例演示了如何使用 `EnableMetricsCollection`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考EnableMetricsCollection中的。

SetDesiredCapacity

以下代码示例演示了如何使用 SetDesiredCapacity。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考SetDesiredCapacity中的。

`TerminateInstanceInAutoScalingGroup`

以下代码示例演示了如何使用 `TerminateInstanceInAutoScalingGroup`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考[TerminateInstanceInAutoScalingGroup](#)中的。

`UpdateAutoScalingGroup`

以下代码示例演示了如何使用 `UpdateAutoScalingGroup`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考[UpdateAutoScalingGroup](#)中的。

使用 Amazon Bedrock 示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Amazon Bedrock 的 适用于 .NET 的 AWS SDK (v4) 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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 操作

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。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 BedrockActions;

/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
```

```
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
 of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-
is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);

    }


    /// <summary>
    /// List foundation models.
    /// </summary>
    /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
    private static async Task ListFoundationModelsAsync(AmazonBedrockClient
 bedrockClient)
    {
        Console.WriteLine("List foundation models with no filter.");

        try
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
 ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }
```

```
    /// <summary>
    /// Write the foundation model summary to console.
    /// </summary>
    /// <param name="foundationModel"> The foundation model summary to write to
 console. </param>
    private static void WriteToConsole(FoundationModelSummary foundationModel)
    {
        Console.WriteLine($"{foundationModel.ModelId}, Customization:
 {string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
 {foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
 ", foundationModel.InputModalities)}, Output: {string.Join(", ",
 foundationModel.OutputModalities)}");
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListFoundationModels中的。

操作

`ListFoundationModels`

以下代码示例演示了如何使用 `ListFoundationModels`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListFoundationModels中的。

使用 Amazon Bedrock 运行时示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Amazon Bedrock Runtime 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI
- Stable Image Core

开始使用

你好 Amazon Bedrock 运行时

以下代码示例展示了如何开始使用 Amazon Bedrock 运行时。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

namespace BedrockRuntimeActions;

/// <summary>
/// This example shows how to use the Converse API to send a text message
/// to Amazon Bedrock.
/// </summary>
internal class HelloBedrockRuntime
{
    static async Task Main(string[] args)
    {
        // Create a Bedrock Runtime client in the AWS Region you want to use.
        // You can change the region to match your setup.
```

```
        var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USWest2);

        // Set the model ID, e.g., Claude Haiku.
        // The "global." prefix enables cross-region inference, allowing the request
        // to be routed to the nearest available region for the specified model.
        var modelId = "global.anthropic.claude-haiku-4-5-20251001-v1:0";

        // Define the user message.
        var userMessage = "Hi. In a short paragraph, explain what you can do.";

        // Create a request with the model ID, the user message, and an inference
 configuration.
        var request = new ConverseRequest
        {
            ModelId = modelId,
            Messages = new List<Message>
            {
                new Message
                {
                    Role = ConversationRole.User,
                    Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
                }
            }
        };

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

            // Extract and print the response text.
            string responseText = response?.Output?.Message?.Content?[0]?.Text ??
 "";
            Console.WriteLine(responseText);
        }
        catch (AmazonBedrockRuntimeException e)
        {
            Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason:
 {e.Message}");
            throw;
        }
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Converse。

Anthropic Claude

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息。

```
// Use the Converse API to send a text message to Anthropic Claude.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
```

```
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Converse。

ConverseStream

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息并实时处理响应流。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Anthropic Claude 发送文本消息并实时处理响应流。

```
// Use the Converse API to send a text message to Anthropic Claude
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
```

```
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ConverseStream中的。

Cohere Command

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Cohere Command 发送文本消息。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Cohere Command 发送文本消息。

```
// Use the Converse API to send a text message to Cohere Command.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Converse。

ConverseStream

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Cohere Command 发送文本消息并实时处理响应流。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Cohere Command 发送文本消息并实时处理响应流。

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

```
    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ConverseStream中的。

Meta Llama

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Meta Llama 发送文本消息。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Meta Llama 发送文本消息。

```
// Use the Converse API to send a text message to Meta Llama.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
```

```
    throw;
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Converse。

ConverseStream

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Meta Llama 发送文本消息并实时处理响应流。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Meta Llama 发送文本消息并实时处理响应流。

```
// Use the Converse API to send a text message to Meta Llama
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";
```

```
// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ConverseStream中的。

Mistral AI

Converse

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Mistral 发送文本消息。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Mistral 发送文本消息。

```
// Use the Converse API to send a text message to Mistral.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
```

```
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Converse。

ConverseStream

以下代码示例演示如何使用 Bedrock 的 Converse API 向 Mistral 发送文本消息并实时处理响应流。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Bedrock 的 Converse API 向 Mistral 发送文本消息并实时处理响应流。

```
// Use the Converse API to send a text message to Mistral
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
```

```
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ConverseStream中的。

Stable Image Core

InvokeModel

以下代码示例展示了如何在 Amazon Bedrock 上调用 St Stability.ai able Image Core 来生成图像。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Stable Image Core 创建镜像。

```
        /// <summary>
        /// Asynchronously invokes the Stability.ai Stable Image Core model to run
 an inference based on the provided input.
        /// </summary>
        /// <param name="prompt">The prompt that describes the image Stability.ai
 Stable Image Core has to generate.</param>
        /// <returns>A base-64 encoded image generated by model</returns>
        /// <remarks>
        /// The different model providers have individual request and response
 formats.
        /// For the format, ranges, and default values for Stability.ai Stable Image
 Core, refer to:
        ///     https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-diffusion-stable-image-core-text-image-request-response.html
        /// </remarks>
        public static async Task<string?> InvokeStableImageCoreAsync(string prompt,
 int seed)
        {
            string stableImageCoreModelId = "stability.stable-image-core-v1:1";

            AmazonBedrockRuntimeClient client = new(RegionEndpoint.USWest2);

            string payload = new JsonObject()
            {
                { "prompt", prompt },
                { "aspect_ratio", "1:1" },
                { "seed", seed },
                { "output_format", "png" }
            }.ToJsonString();

            try
            {
```

```
                InvokeModelResponse response = await client.InvokeModelAsync(new
 InvokeModelRequest()
                {
                    ModelId = stableImageCoreModelId,
                    Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
                    ContentType = "application/json",
                    Accept = "application/json"
                });

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    var results = JsonNode.ParseAsync(response.Body).Result?
["images"]?.AsArray();

                    return results?[0]?.GetValue<string>();
                }
                else
                {
                    Console.WriteLine("InvokeModelAsync failed with status code " +
 response.HttpStatusCode);
                }
            }
            catch (AmazonBedrockRuntimeException e)
            {
                Console.WriteLine(e.Message);
            }
            return null;
        }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考InvokeModel中的。

CloudFormation 使用示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何通过使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。CloudFormation

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用

开始使用

Hello CloudFormation

以下代码示例展示了如何开始使用 CloudFormation。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

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

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
```

```
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            if (paginatorForDescribeStacks.Stacks != null)
            {
                await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
                {
                    // Basic information for each stack
                    Console.WriteLine(
                        "\n-----------------------------------------------");
                    Console.WriteLine($"\nStack: {stack.StackName}");
                    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                    Console.WriteLine($"  Created: {stack.CreationTime}");

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

                    // The resources of each stack
                    DescribeStackResourcesResponse responseDescribeResources =
                        await _amazonCloudFormation.DescribeStackResourcesAsync(
                            new DescribeStackResourcesRequest
                            {
                                StackName = stack.StackName
                            });
                    if (responseDescribeResources.StackResources != null &&
responseDescribeResources.StackResources.Count > 0)
                    {
                        Console.WriteLine("  Resources:");
                        foreach (StackResource resource in responseDescribeResources
                                     .StackResources)
                            Console.WriteLine(
                                $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
```

```
                    }
                }
            }

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

            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeStackResources中的。

CloudWatch 使用示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何通过使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。CloudWatch

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

你好 CloudWatch

以下代码示例展示了如何开始使用 CloudWatch。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m =>
 $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
```

```
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListMetrics中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 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。
- 创建并触发警报，然后查看警报历史记录。
- 添加异常检测器
- 获取指标映像，然后清理资源。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

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

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

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

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");
```

```
        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

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

    /// <summary>
    /// Get and display estimated billing statistics.
```

```
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

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

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
```

```
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();
```

```
        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

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

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

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


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();
```

```
        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
```

```
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
```

```
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
```

```
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

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

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];
```

```
        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

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

```
    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

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

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
```

```
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");

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

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
```

```
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

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

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

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

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

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

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
```

```
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

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

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
```

```
                anomalyDetector);
        }

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

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

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 ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }
```

```
    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
```

```
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
```

```
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
```

```
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
```

```
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
```

```
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
```

```
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
```

```
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
```

```
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

场景的 settings.json 值示例。

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
```

```
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
example code.\n"
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
            "left": {
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
```

```
        "type": "metric",
        "properties": {
          "metrics": [
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
 "Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      }
    ]
  }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric

- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

操作

`DeleteAlarms`

以下代码示例演示了如何使用 `DeleteAlarms`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DeleteAlarms中的。

`DeleteAnomalyDetector`

以下代码示例演示了如何使用 `DeleteAnomalyDetector`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DeleteAnomalyDetector中的。

DeleteDashboards

以下代码示例演示了如何使用 DeleteDashboards。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DeleteDashboards中的。

DescribeAlarmHistory

以下代码示例演示了如何使用 DescribeAlarmHistory。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeAlarmHistory中的。

`DescribeAlarms`

以下代码示例演示了如何使用 `DescribeAlarms`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeAlarms中的。

`DescribeAlarmsForMetric`

以下代码示例演示了如何使用 `DescribeAlarmsForMetric`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 ?? new List<MetricAlarm>();
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeAlarmsForMetric中的。

`DescribeAnomalyDetectors`

以下代码示例演示了如何使用 `DescribeAnomalyDetectors`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DescribeAnomalyDetectors中的。

DisableAlarmActions

以下代码示例演示了如何使用 DisableAlarmActions。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考DisableAlarmActions中的。

EnableAlarmActions

以下代码示例演示了如何使用 EnableAlarmActions。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考EnableAlarmActions中的。

`GetDashboard`

以下代码示例演示了如何使用 `GetDashboard`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考GetDashboard中的。

`GetMetricData`

以下代码示例演示了如何使用 `GetMetricData`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetMetricData中的。

GetMetricStatistics

以下代码示例演示了如何使用 GetMetricStatistics。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 ?? new List<Datapoint>();
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetMetricStatistics中的。

`GetMetricWidgetImage`

以下代码示例演示了如何使用 `GetMetricWidgetImage`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考GetMetricWidgetImage中的。

`ListDashboards`

以下代码示例演示了如何使用 `ListDashboards`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考ListDashboards中的。

`ListMetrics`

以下代码示例演示了如何使用 `ListMetrics`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考ListMetrics中的。

PutAnomalyDetector

以下代码示例演示了如何使用 PutAnomalyDetector。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考PutAnomalyDetector中的。

PutDashboard

以下代码示例演示了如何使用 PutDashboard。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 ?? new
 List<DashboardValidationMessage>();
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考PutDashboard中的。

PutMetricAlarm

以下代码示例演示了如何使用 PutMetricAlarm。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考PutMetricAlarm中的。

PutMetricData

以下代码示例演示了如何使用 PutMetricData。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考PutMetricData中的。

CloudWatch 使用记录示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用 适用于 .NET 的 AWS SDK (v4) with Lo CloudWatch gs 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的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从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

`GetQueryResults`

以下代码示例演示了如何使用 `GetQueryResults`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 results of a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="queryId">The ID of the query.</param>
    /// <returns>The query results response.</returns>
    public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
    {
        try
        {
            var request = new GetQueryResultsRequest
            {
                QueryId = queryId
            };

            var response = await
_amazonCloudWatchLogs.GetQueryResultsAsync(request);
            return response;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"An error occurred while getting query results:
 {ex.Message}");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetQueryResults中的。

`StartQuery`

以下代码示例演示了如何使用 `StartQuery`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
 epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
```

```
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
{ex.Message}");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考StartQuery中的。

场景

运行大型查询

以下代码示例展示了如何使用 CloudWatch 日志查询超过 10,000 条记录。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

这是演示大型查询场景的主要工作流程。

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:

    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
```

```
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();

            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
 host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();
```

```
        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);

                        Console.WriteLine("Application preparation complete.");
                        return true;
                    }
                }
            }
            else
            {
```

```
                _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
                _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

                var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
                var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

                _queryStartDate = startDateMs / 1000;
                _queryEndDate = endDateMs / 1000;

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }

        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };
```

```
            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
```

```
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
```

```
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
                    logEvents.Add(new InputLogEvent
                    {
                        Timestamp =
DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }
```

```
                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
    /// Executes the large query workflow.
    /// </summary>
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
 10000);
        if (queryLimit > 10000) queryLimit = 10000;
```

```
        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
        }
        Console.WriteLine();

        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
```

```
                var timestamp = allResults[i].Find(f => f.Field ==
 "@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
 "@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
 startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
 DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
```

```
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");
```

```
        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
 range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");

        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
 range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);
```

```
        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
```

```
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }

    /// <summary>
    /// Cleans up the resources created during the scenario.
    /// </summary>
    public static async Task<bool> Cleanup()
    {
        var cleanup = !_interactive || GetYesNoResponse(
            "Do you want to delete the CloudFormation stack and all resources? (y/n)
");
```

```
        if (cleanup)
        {
            try
            {
                var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
                return stackDeleteSuccess;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
                return false;
            }
        }

        Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
        _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
        return true;
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
    {
        var request = new DeleteStackRequest
        {
            StackName = stackName,
        };

        if (forceDelete)
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);
```

```
        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
```

```
                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
        }
        Console.WriteLine("\rWait complete.                      ");
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
```

```
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }
```

```
    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }

    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();
```

```
        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - GetQueryResults
  - StartQuery

使用 Amazon Cognito 身份提供商示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Amazon Cognito 身份提供商的 适用于 .NET 的 AWS SDK (v4) 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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从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

`ListUserPools`

以下代码示例演示了如何使用 `ListUserPools`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 适用于 .NET 的 AWS SDK API 参考ListUserPools中的。

AWS Control Tower 使用示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何通过使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。AWS Control Tower

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

Hello AWS Control Tower

以下代码示例展示了如何开始使用 AWS Control Tower。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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.ControlTower;
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloControlTower>();

        var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

        Console.Clear();
        Console.WriteLine("Hello, AWS Control Tower! Let's list available
 baselines:");
        Console.WriteLine();

        var baselines = new List<BaselineSummary>();

        try
        {
```

```
            var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
 ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
            foreach (var baseline in baselines)
            {
                Console.WriteLine($"\t{baseline.Name}");
            }
        }
        catch (Amazon.ControlTower.Model.AccessDeniedException)
        {
            Console.WriteLine("Access denied. Please ensure you have the necessary
 permissions.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListBaselines中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 列出登录区。
- 列出、启用、获取、重置和禁用基准。
- 列出、启用、获取和禁用控件。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行演示 AWS Control Tower 功能的交互式场景。

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
```

```
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
GetCallerIdentityRequest())).Account;
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
these operations in the demo.");
```

```
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }

            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;

            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
```

```
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
 controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
                                    alreadyEnabled = true;
                                    Console.WriteLine("No change, the selected
 baseline was already enabled.");
                                    break;
                                }
                            }
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
 Baseline? (y/n): ");
```

```
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
                                    // Re-enable the Control Tower baseline if it
was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "5.0",
icBaselineArn ?? "");
                                }
                            }
                        }
                    }
                }
            }

            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
{controls[i].Arn}");
```

```
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");

                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
 wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
 {disableOpId}");
                            }
                        }
                    }
```

```
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;
```

```
        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
 ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
 "Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
 CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
    /// Gets user confirmation by waiting for input or returning true if not
 interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
 false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
}
```

场景调用以管理 Aurora 操作的包装程序方法。

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
```

```
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
 _controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }
```

```
            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
```

```
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
 _controlTowerService.Paginators.ListEnabledBaselines(new
 ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
```

```
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
```

```
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
```

```
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
```

```
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
```

```
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
 (ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
 enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
```

```
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }
```

```
            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            var request = new DisableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.DisableControlAsync(request);
```

```
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
```

```
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();

            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
        catch (AmazonControlCatalogException ex)
        {
```

```
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
 {ex.Message}");
            throw;
        }
    }

}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

操作

`DisableBaseline`

以下代码示例演示了如何使用 `DisableBaseline`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a baseline for a specific target and wait for the operation to
complete.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
   /// <returns>The operation ID or null if there was a conflict.</returns>
   public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new DisableBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await _controlTowerService.DisableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
```

```
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DisableBaseline中的。

`DisableControl`

以下代码示例演示了如何使用 `DisableControl`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
    {
```

```
        try
        {
            var request = new DisableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.DisableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DisableControl中的。

EnableBaseline

以下代码示例演示了如何使用 EnableBaseline。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
```

```
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考EnableBaseline中的。

EnableControl

以下代码示例演示了如何使用 EnableControl。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to enable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID or null if already enabled.</returns>
   public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           Console.WriteLine(controlArn);
           Console.WriteLine(targetIdentifier);

           var request = new EnableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.EnableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
```

```
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考EnableControl中的。

GetBaselineOperation

以下代码示例演示了如何使用 GetBaselineOperation。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[GetBaselineOperation](#)中的。

GetControlOperation

以下代码示例演示了如何使用 GetControlOperation。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 status of a control operation.
   /// </summary>
   /// <param name="operationId">The ID of the control operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetControlOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetControlOperationAsync(request);
           return response.ControlOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetControlOperation中的。

`ListBaselines`

以下代码示例演示了如何使用 `ListBaselines`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListBaselines中的。

`ListEnabledBaselines`

以下代码示例演示了如何使用 `ListEnabledBaselines`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
```

```
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListEnabledBaselines中的。

`ListEnabledControls`

以下代码示例演示了如何使用 `ListEnabledControls`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
```

```
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListEnabledControls中的。

`ListLandingZones`

以下代码示例演示了如何使用 `ListLandingZones`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
```

```
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListLandingZones中的。

`ResetEnabledBaseline`

以下代码示例演示了如何使用 `ResetEnabledBaseline`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
```

```
   /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new ResetEnabledBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Target not found, unable to reset enabled
baseline.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
```

```
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ResetEnabledBaseline中的。

使用 DynamoDB 示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 DynamoDB 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 DynamoDB

以下代码示例显示如何开始使用 DynamoDB。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
```

```
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListTables中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 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。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
```

```
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
```

```
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous" +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
```

```
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
```

```
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using DynamoDB
with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
```

```
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
 given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
 a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

使用注入的客户端进行表操作。

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
```

```
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

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="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
```

```
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
```

```
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
 {ex.Message}");
            throw;
        }
    }
```

将单个影片添加到表中。

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };
```

```
            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

更新表中的单个项目。

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
```

```
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

从影片表中检索单个项目。

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

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 name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
```

```
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();
```

```
            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

从表中删除单个项目。

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
```

```
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

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="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
```

```
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);
```

```
            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

扫描此表以查找几年内发行的影片。

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
```

```
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
```

```
        }
    }
```

删除影片表。

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

操作

`BatchWriteItem`

以下代码示例演示了如何使用 `BatchWriteItem`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
```

```
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考BatchWriteItem中的。

`CreateTable`

以下代码示例演示了如何使用 `CreateTable`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
```

```
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

            return status == TableStatus.ACTIVE;
        }
```

```
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateTable中的。

`DeleteItem`

以下代码示例演示了如何使用 `DeleteItem`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="tableName">The name of the table from which the item
    /// will be deleted.</param>
```

```
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteItem中的。

`DeleteTable`

以下代码示例演示了如何使用 `DeleteTable`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
```

```
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteTable中的。

`GetItem`

以下代码示例演示了如何使用 `GetItem`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
```

```
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetItem中的。

PutItem

以下代码示例演示了如何使用 PutItem。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考PutItem中的。

Query

以下代码示例演示了如何使用 Query。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库 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="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
```

```
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Query。

Scan

以下代码示例演示了如何使用 Scan。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
```

```
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
 table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
 {ex.Message}");
            throw;
        }
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 Scan。

`UpdateItem`

以下代码示例演示了如何使用 `UpdateItem`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
```

```
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考UpdateItem中的。

使用 Amazon EC2 的示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何在 Amazon EC2 中使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用

开始使用

开始使用 Amazon EC2

以下代码示例显示了如何开始使用 Amazon EC2。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
 EC2).
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                .AddTransient<EC2Wrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);

            await foreach (var securityGroup in
 paginatorForSecurityGroups.SecurityGroups)
            {
                securityGroups.Add(securityGroup);
            }

            // Now print the security groups returned by the call to
            // DescribeSecurityGroupsAsync.
            Console.WriteLine("Welcome to the EC2 Hello Service example. " +
```

```
                                "\nLet's list your Security Groups:");
            securityGroups.ForEach(group =>
            {
                Console.WriteLine(
                    $"Security group: {group.GroupName} ID: {group.GroupId}");
            });
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"An Amazon EC2 service error occurred while listing
 security groups. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing security groups.
 {ex.Message}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeSecurityGroups中的。

使用 Amazon ECS 示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Amazon ECS 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用

开始使用

Hello Amazon ECS

以下代码示例展示了如何开始使用 Amazon ECS。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon ECS client.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonECS>()
            )
            .Build();

        var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();
```

```
        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your account");
        Console.WriteLine();

        var clusters = new List<string>();

        var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
 ListClustersRequest());

        await foreach (var response in clustersPaginator.Responses)
        {
            clusters.AddRange(response.ClusterArns);
        }

        if (clusters.Count > 0)
        {
            clusters.ForEach(cluster =>
            {
                Console.WriteLine($"\tARN: {cluster}");
                Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
                Console.WriteLine();
            });
        }
        else
        {
            Console.WriteLine("No clusters were found.");
        }

    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListClusters中的。

AWS IoT 使用示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何通过使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。AWS IoT

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

Hello AWS IoT

以下代码示例展示了如何开始使用 AWS IoT。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
```

```
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
```

```
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的 listThings。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建 AWS IoT 事物。
- 生成设备证书。
- 使用属性更新 AWS IoT 事物。

- 返回一个唯一的端点。
- 列出您的 AWS IoT 证书。
- 更新 AWS IoT 阴影。
- 写出状态信息。
- 创建规则。
- 列出您的规则。
- 使用事物名称搜索事物。
- 删除 AWS IoT 事物。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行演示 AWS IoT 功能的交互式场景。

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
```

```
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
```

```
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
```

```
   /// <returns>A Task object.</returns>
   private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       string thingName = $"iot-thing-{Guid.NewGuid():N}";
       string certificateArn = "";
       string certificateId = "";
       string ruleName = $"iotruledefault";
       string snsTopicArn = "";

       try
       {
           // Step 1: Create an AWS IoT Thing
           Console.WriteLine(new string('-', 80));
           Console.WriteLine("1. Create an AWS IoT Thing.");
           Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
AWS IoT service that can be associated with a physical device.");
           Console.WriteLine();

           if (IsInteractive)
           {
               Console.Write("Enter Thing name: ");
               var userInput = Console.ReadLine();
               if (!string.IsNullOrEmpty(userInput))
                   thingName = userInput;
           }
           else
           {
               Console.WriteLine($"Using default Thing name: {thingName}");
           }

           var thingArn = await iotWrapper.CreateThingAsync(thingName);
           Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
           Console.WriteLine(new string('-', 80));

           // Step 1.1: List AWS IoT Things
           Console.WriteLine(new string('-', 80));
           Console.WriteLine("2. List AWS IoT Things.");
           Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
           Console.WriteLine();
           if (IsInteractive)
           {
```

```
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
```

```
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
                    {
                        Console.WriteLine("Certificate attached to Thing
successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Failed to attach certificate to Thing.");
                    }

                    Console.WriteLine("Thing Details:");
                    Console.WriteLine($"Thing Name: {thingName}");
                    Console.WriteLine($"Thing ARN: {thingArn}");
                }
                else
                {
```

```
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var attributes = new Dictionary<string, string>
        {
            { "Location", "Seattle" },
            { "DeviceType", "Sensor" },
            { "Firmware", "1.2.3" }
        };

        await iotWrapper.UpdateThingAsync(thingName, attributes);
        Console.WriteLine("Thing attributes updated successfully.");
        Console.WriteLine(new string('-', 80));

        // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var endpoint = await iotWrapper.DescribeEndpointAsync();
        if (endpoint != null)
```

```
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
```

```
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
```

```
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
```

```
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and rule
creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
```

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

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the certificate
 for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
 {thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
```

```
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
```

```
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
 resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
 delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
 iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
 logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
```

```
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
```

```
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
```

```
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);
```

```
            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
```

```
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
```

```
            Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
```

```
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

```
    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
 for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

AWS IoT SDK 方法的包装器类。

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
```

```
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
```

```
   /// Creates a device certificate for AWS IoT.
   /// </summary>
   /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
           var request = new CreateKeysAndCertificateRequest
           {
               SetAsActive = true
           };

           var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
           _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
           return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
       {
           _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
           return null;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
           return null;
       }
   }

   /// <summary>
   /// Attaches a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
```

```
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
```

```
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
```

```
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };
```

```
            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
```

```
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
```

```
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }
```

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
```

```
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
```

```
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
```

```
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
```

```
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
```

```
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
```

```
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
 page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule

- DescribeEndpoint
- DescribeThing
- DetachThingPrincipal
- ListCertificates
- ListThings
- SearchIndex
- UpdateIndexingConfiguration
- UpdateThing

操作

`AttachThingPrincipal`

以下代码示例演示了如何使用 `AttachThingPrincipal`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
```

```
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考AttachThingPrincipal中的。

`CreateKeysAndCertificate`

以下代码示例演示了如何使用 `CreateKeysAndCertificate`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
```

```
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
           var request = new CreateKeysAndCertificateRequest
           {
               SetAsActive = true
           };

           var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
           _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
           return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
       {
           _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
           return null;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
           return null;
       }
   }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[CreateKeysAndCertificate](#)中的。

CreateThing

以下代码示例演示了如何使用 CreateThing。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateThing中的。

CreateTopicRule

以下代码示例演示了如何使用 CreateTopicRule。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                    },
                    RuleDisabled = false
```

```
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateTopicRule中的。

`DeleteCertificate`

以下代码示例演示了如何使用 `DeleteCertificate`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
```

```
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteCertificate中的。

DeleteThing

以下代码示例演示了如何使用 DeleteThing。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteThing中的。

`DescribeEndpoint`

以下代码示例演示了如何使用 `DescribeEndpoint`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeEndpoint中的。

`DetachThingPrincipal`

以下代码示例演示了如何使用 `DetachThingPrincipal`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 certificate from an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to detach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new DetachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.DetachThingPrincipalAsync(request);
           _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
```

```
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DetachThingPrincipal中的。

`ListCertificates`

以下代码示例演示了如何使用 `ListCertificates`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
```

```
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListCertificates中的。

`ListThings`

以下代码示例演示了如何使用 `ListThings`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);
```

```
            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListThings中的。

SearchIndex

以下代码示例演示了如何使用 SearchIndex。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
   /// Searches for IoT Things using the search index.
   /// </summary>
   /// <param name="queryString">The search query string.</param>
   /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
   public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
   {
       try
       {
           // First, try to perform the search
           var request = new SearchIndexRequest
           {
               QueryString = queryString
           };

           var response = await _amazonIoT.SearchIndexAsync(request);
           _logger.LogInformation($"Search found {response.Things.Count} Things");
           return response.Things;
       }
       catch (Amazon.IoT.Model.IndexNotReadyException ex)
       {
           _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
           return await SetupIndexAndRetrySearchAsync(queryString);
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
       {
           _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
           return await SetupIndexAndRetrySearchAsync(queryString);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
```

```
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
```

```
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }
```

```
            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考SearchIndex中的。

`UpdateThing`

以下代码示例演示了如何使用 UpdateThing。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
```

```
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考UpdateThing中的。

AWS IoT data 使用示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何通过使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。AWS IoT data

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从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

`GetThingShadow`

以下代码示例演示了如何使用 `GetThingShadow`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetThingShadow中的。

UpdateThingShadow

以下代码示例演示了如何使用 UpdateThingShadow。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
```

```
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考UpdateThingShadow中的。

使用 Amazon Redshift 示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何使用带有 Amazon Redshift 的 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作

开始使用

开始使用 Amazon Redshift

以下代码示例展示了如何开始使用 Amazon Redshift。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
```

```
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeClusters中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建 Redshift 集群。
- 列出集群中的数据库。
- 创建名为 Movies 的文件。
- 填充 Movies 表。
- 按年份查询 Movies 表。
- 修改 Redshift 集群。
- 删除 Amazon Redshift 集群。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Redshift 包装器类来管理操作。

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
 IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
 databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
 "ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
```

```
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
```

```
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
 {cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };
```

```
            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
    {
        try
        {
            var sqlStatement = @"
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
```

```
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
```

```
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

```
    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);
```

```
        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
 cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

运行一个交互式场景，演示 Redshift 的基础功能。

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
```

```
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
```

```
                    userName = userInput;


 Console.WriteLine("=============================================================================
                Console.WriteLine("Please enter a user password for the cluster
 (default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


 Console.WriteLine("=============================================================================

                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
 clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
 userName, userPassword);

 Console.WriteLine("=============================================================================

            // Step 4: Wait for cluster to become available

 Console.WriteLine("=============================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

 Console.WriteLine("=============================================================================

            // Step 5: List databases

 Console.WriteLine("=============================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
 database is created by default and used in this scenario.");
```

```
            Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

Console.WriteLine("=========================================================================


Console.WriteLine("=========================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("=========================================================================

            // Step 6: Create Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("=========================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
                    recordCount = inputCount;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("=============================================================================

            // Step 8 & 9: Query movies by year

Console.WriteLine("=============================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)
```

```
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
 {year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
 userName, year);

 Console.WriteLine("=============================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=============================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=============================================================================

            // Step 11 & 12: Delete cluster confirmation

 Console.WriteLine("=============================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
 cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
```

```
                    await
Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

Console.WriteLine("=============================================================================


Console.WriteLine("=============================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=============================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
        }
```

```
        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
 file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult

- ListDatabasesPaginator
- ModifyCluster

操作

`CreateCluster`

以下代码示例演示了如何使用 `CreateCluster`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Redshift cluster.
   /// </summary>
   /// <param name="clusterIdentifier">The identifier for the cluster.</param>
   /// <param name="databaseName">The name of the database.</param>
   /// <param name="masterUsername">The master username.</param>
   /// <param name="masterUserPassword">The master user password.</param>
   /// <param name="nodeType">The node type for the cluster.</param>
   /// <returns>The cluster that was created.</returns>
   public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
       string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
   {
       try
       {
           var request = new CreateClusterRequest
           {
               ClusterIdentifier = clusterIdentifier,
               DBName = databaseName,
               MasterUsername = masterUsername,
               MasterUserPassword = masterUserPassword,
               NodeType = nodeType,
               NumberOfNodes = 1,
```

```
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateCluster中的。

`DeleteCluster`

以下代码示例演示了如何使用 `DeleteCluster`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
```

```
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteCluster中的。

`DescribeClusters`

以下代码示例演示了如何使用 `DescribeClusters`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 Amazon Redshift clusters.
    /// </summary>
```

```
   /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
   /// <returns>A list of clusters.</returns>
   public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
   {
       try
       {
           var clusters = new List<Cluster>();
           var request = new DescribeClustersRequest();
           if (!string.IsNullOrEmpty(clusterIdentifier))
           {
               request.ClusterIdentifier = clusterIdentifier;
           }

           var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
           await foreach (var response in clustersPaginator.Responses)
           {
               if (response.Clusters != null)
                   clusters.AddRange(response.Clusters);
           }

           Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
           foreach (var cluster in clusters)
           {
               Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
           }

           return clusters;
       }
       catch (ClusterNotFoundException ex)
       {
           Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
           Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
           throw;
       }
```

```
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeClusters中的。

`DescribeStatement`

以下代码示例演示了如何使用 `DescribeStatement`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 a statement execution.
   /// </summary>
   /// <param name="statementId">The statement ID.</param>
   /// <returns>The statement description.</returns>
   public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
   {
       try
       {
           var request = new DescribeStatementRequest
           {
               Id = statementId
           };

           var response = await
_redshiftDataClient.DescribeStatementAsync(request);
           return response;
       }
       catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
       {
           Console.WriteLine($"Statement not found: {ex.Message}");
           throw;
       }
       catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeStatement中的。

`GetStatementResult`

以下代码示例演示了如何使用 `GetStatementResult`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetStatementResult中的。

`ListDatabases`

以下代码示例演示了如何使用 `ListDatabases`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
```

```
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[ListDatabases](#)中的。

`ModifyCluster`

以下代码示例演示了如何使用 `ModifyCluster`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
```

```
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
 has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ModifyCluster中的。

使用 Amazon S3 的示例 适用于 .NET 的 SDK (v4)

以下代码示例向您展示了如何在 Amazon S3 中使用 适用于 .NET 的 AWS SDK (v4) 来执行操作和实现常见场景。

基本功能是向您展示如何在服务中执行基本操作的代码示例。

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的上下文查看操作。

场景是向您演示如何通过在一个服务中调用多个函数或与其他 AWS 服务结合来完成特定任务的代码示例。

每个示例都包含一个指向完整源代码的链接，您可以从中找到有关如何在上下文中设置和运行代码的说明。

主题

- 开始使用
- 基本功能
- 操作
- 场景

开始使用

开始使用 Amazon S3

以下代码示例显示了如何开始使用 Amazon S3。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
/// Hello Amazon Simple Storage Service
// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
```

```
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var s3Client = new AmazonS3Client();

        try
        {
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));

            // Use the built-in paginator to list buckets
            var request = new ListBucketsRequest();
            var paginator = s3Client.Paginators.ListBuckets(request);

            var buckets = new List<S3Bucket>();

            await foreach (var response in paginator.Responses)
            {
                buckets.AddRange(response.Buckets);
            }

            if (buckets.Any())
            {
                Console.WriteLine($"Found {buckets.Count} S3 buckets:");
                Console.WriteLine();

                foreach (var bucket in buckets)
                {
                    Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
                    Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-
MM-dd HH:mm:ss UTC}");
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No S3 buckets found in your account.");
            }

            Console.WriteLine("Hello S3 completed successfully.");
        }
        catch (AmazonS3Exception ex)
        {
```

```
            Console.WriteLine($"S3 service error occurred: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list S3 buckets. Here's why:
 {ex.Message}");
        }
    }
}
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListBuckets中的。

基本功能

了解基本功能

以下代码示例展示了如何：

- 创建桶并将文件上载到其中。
- 从桶中下载对象。
- 将对象复制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。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行一个交互式场景，演示 Amazon S3 的功能。

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
```

```
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddLogging(builder => builder.AddConsole()))
            .Build();

        _logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<S3_Basics>();

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

        var sepBar = new string('-', 45);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
        Console.WriteLine("\n\t5. Delete all the items in the bucket");
        Console.WriteLine("\n\t6. Delete the bucket");
        Console.WriteLine(sepBar);

        await RunScenario(_s3Wrapper, _logger);

        Console.WriteLine(sepBar);
        Console.WriteLine("The Amazon S3 scenario has successfully completed.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Run the S3 Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="s3Wrapper">The S3 wrapper instance.</param>
```

```
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
    {
        string bucketName = BucketName;
        string filePath = TempFilePath;
        string keyName = string.Empty;

        var sepBar = new string('-', 45);

        try
        {
            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                Console.Write("Please enter a name for the new bucket: ");
                bucketName = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Using bucket name: {bucketName}");
            }

            var success = await s3Wrapper.CreateBucketAsync(bucketName);
            if (success)
            {
                Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not create bucket: {bucketName}.\n");
            }

            Console.WriteLine(sepBar);
            Console.WriteLine("Upload a file to the new bucket.");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
```

```
                // Get the local path and filename for the file to upload.
                while (string.IsNullOrEmpty(filePath))
                {
                    Console.Write("Please enter the path and filename of the file to
 upload: ");
                    filePath = Console.ReadLine();

                    // Confirm that the file exists on the local computer.
                    if (!File.Exists(filePath))
                    {
                        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
                        filePath = string.Empty;
                    }
                }
            }
            else
            {
                // Use the public variable if set, otherwise create a temp file
                if (!string.IsNullOrEmpty(TempFilePath))
                {
                    filePath = TempFilePath;
                    Console.WriteLine($"Using provided test file: {filePath}");
                }
                else
                {
                    // Create a temporary test file for non-interactive mode
                    filePath = Path.GetTempFileName();
                    var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
                    await File.WriteAllTextAsync(filePath, testContent);
                    Console.WriteLine($"Created temporary test file: {filePath}");
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
 filePath);

            if (success)
            {
```

```
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set up download path
            string downloadPath = string.Empty;

            if (IsInteractive)
            {
                // Now get a new location where we can save the file.
                while (string.IsNullOrEmpty(downloadPath))
                {
                    // First get the path to which the file will be downloaded.
                    Console.Write("Please enter the path where the file will be
downloaded: ");
                    downloadPath = Console.ReadLine();

                    // Confirm that the file doesn't already exist on the local
computer.
                    if (File.Exists($"{downloadPath}\\{keyName}"))
                    {
                        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                        downloadPath = string.Empty;
                    }
                }
            }
            else
            {
                downloadPath = Path.GetTempPath();
                var downloadFile = Path.Combine(downloadPath, keyName);
                if (File.Exists(downloadFile))
                {
                    File.Delete(downloadFile);
                }

                Console.WriteLine($"Using download path: {downloadPath}");
            }

            // Download an object from a bucket.
```

```
            success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);

            if (success)
            {
                Console.WriteLine($"Successfully downloaded {keyName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not download {keyName}.\n");
            }

            // Copy the object to a different folder in the bucket.
            string folderName = string.Empty;

            if (IsInteractive)
            {
                while (string.IsNullOrEmpty(folderName))
                {
                    Console.Write("Please enter the name of the folder to copy your
object to: ");
                    folderName = Console.ReadLine();
                }
            }
            else
            {
                folderName = "test-folder";
                Console.WriteLine($"Using folder name: {folderName}");
            }

            await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
folderName);

            // List the objects in the bucket.
            await s3Wrapper.ListBucketContentsAsync(bucketName);

            // Delete the contents of the bucket.
            if (IsInteractive)
            {
                Console.WriteLine("Press <Enter> when you are ready to delete the
bucket contents.");
                _ = Console.ReadLine();
            }
```

```
            var deleteContentsSuccess = await
 s3Wrapper.DeleteBucketContentsAsync(bucketName);
            if (deleteContentsSuccess)
            {
                Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete contents of
 {bucketName}.\n");
            }

            if (IsInteractive)
            {
                // Deleting the bucket too quickly after separately deleting its
 contents can
                // cause an error that the bucket isn't empty. To delete contents
 and bucket in one
                // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
                Console.WriteLine("Press <Enter> when you are ready to delete the
 bucket.");
                _ = Console.ReadLine();
            }
            else
            {
                // Add a small delay for non-interactive mode to ensure objects are
 fully deleted.
                Console.WriteLine("Waiting a moment for objects to be fully
 deleted...");
                await Task.Delay(2000);
            }

            // Delete the bucket.
            var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
            if (deleteSuccess)
            {
                Console.WriteLine($"Successfully deleted {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
            }
```

```
            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (File.Exists(filePath))
                    {
                        File.Delete(filePath);
                        Console.WriteLine("Cleaned up temporary test file.");
                    }

                    var downloadFile = Path.Combine(downloadPath, keyName);
                    if (File.Exists(downloadFile))
                    {
                        File.Delete(downloadFile);
                        Console.WriteLine("Cleaned up downloaded test file.");
                    }
                }
                catch (Exception ex)
                {
                    scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
                }
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

            // Clean up on error - delete bucket if it exists
            try
            {
                if (!string.IsNullOrEmpty(bucketName))
                {
                    await s3Wrapper.DeleteBucketContentsAsync(bucketName);
                    await s3Wrapper.DeleteBucketAsync(bucketName);
                }
            }
            catch (Exception cleanupEx)
            {
                scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
            }
```

```
            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
                    {
                        File.Delete(filePath);
                    }
                }
                catch (Exception fileCleanupEx)
                {
                    scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
 temporary files during error handling.");
                }
            }

            throw;
        }
    }
}
```

Amazon S3 SDK 方法的包装器类。

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

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
```

```
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
    }


    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
```

```
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        var request = new GetObjectRequest
```

```
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Copies an object in an Amazon S3 bucket to a folder within the
    /// same bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object to copy is located.</param>
    /// <param name="objectName">The object to be copied.</param>
    /// <param name="folderName">The folder to which the object will
    /// be copied.</param>
    /// <returns>A boolean value that indicates the success or failure of
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
```

```
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
```

```
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }



    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
```

```
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest { BucketName = bucketName, };

            await _amazonS3.DeleteBucketAsync(request);
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting bucket: {ex.Message}");
            return false;
        }
    }
```

```
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API Reference》中的以下主题。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

操作

`CopyObject`

以下代码示例演示了如何使用 `CopyObject`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
```

```
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[CopyObject](#)中的。

`CreateBucket`

以下代码示例演示了如何使用 `CreateBucket`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)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="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateBucket中的。

`CreatePresignedPost`

以下代码示例演示了如何使用 `CreatePresignedPost`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建预签名 POST URL。

```
    /// <summary>
    /// Create a presigned POST URL with conditions.
    /// </summary>
    /// <param name="s3Client">The Amazon S3 client.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
 be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
```

```
            }
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreatePresignedPost中的。

`DeleteBucket`

以下代码示例演示了如何使用 `DeleteBucket`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
```

```
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteBucket中的。

`DeleteObjects`

以下代码示例演示了如何使用 `DeleteObjects`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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 all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }
```

```
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteObjects中的。

`GetObject`

以下代码示例演示了如何使用 `GetObject`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
```

```
    {
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetObject中的。

`ListObjectsV2`

以下代码示例演示了如何使用 `ListObjectsV2`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }
```

```
            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考中的 ListObjectsV2。

`PutObject`

以下代码示例演示了如何使用 `PutObject`。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代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="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
```

```
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考PutObject中的。

场景

创建预签名 URL

以下代码示例展示了如何为 Amazon S3 创建预签名 URL 以及如何上传对象。

适用于 .NET 的 SDK (v4)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建并使用预签名 POST URL 来进行直接浏览器上传。

```
/// <summary>
```

```
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }

    /// <summary>
    /// Perform the actions defined for the Amazon S3 Presigned POST scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        _isInteractive = !args.Contains("--non-interactive");

        // Set up dependency injection for Amazon S3
```

```
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddHttpClient()
            )
            .Build();

        SetUpServices(host);

        try
        {
            // Display overview
            _uiMethods.DisplayOverview();
            _uiMethods.PressEnter(_isInteractive);

            // Step 1: Create bucket
            await CreateBucketAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 2: Create presigned URL
            _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
            var response = await CreatePresignedPostAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 3: Display URL and fields
            _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
            DisplayPresignedPostFields(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 4: Upload file
            _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
URL");
            await UploadFileAsync(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 5: Verify file exists
            await VerifyFileExistsAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 6: Cleanup
            _uiMethods.DisplayTitle("Step 6: Clean up resources");
            await CleanupAsync();
```

```
            _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
 successfully!");
            _uiMethods.PressEnter(_isInteractive);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error in scenario");
            Console.WriteLine($"Error: {ex.Message}");

            // Attempt cleanup if there was an error
            if (!string.IsNullOrEmpty(_bucketName))
            {
                _uiMethods.DisplayTitle("Cleaning up resources after error");
                await _s3Wrapper.DeleteBucketAsync(_bucketName);
                Console.WriteLine($"Cleaned up bucket: {_bucketName}");
            }
        }
    }

    /// <summary>
    /// Create an S3 bucket for the scenario.
    /// </summary>
    private static async Task CreateBucketAsync()
    {
        _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

        // Generate a default bucket name for the scenario
        var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

        // Prompt user for bucket name or use default in non-interactive mode
        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}'): ",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrWhiteSpace(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");
```

```
        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }


    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
    private static async Task<CreatePresignedPostResponse>
 CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
 demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);

        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.
    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
 scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");
```

```
        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
{uploadResult.StatusCode}");
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
{uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {
        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
```

```
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);

            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
 formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
 httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
 responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>
    private static async Task VerifyFileExistsAsync()
    {
        _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

        Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

        try
        {
            var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
 _objectKey!);
```

```
            Console.WriteLine($"File verification successful! File exists in the
bucket.");
            Console.WriteLine($"File size: {metadata.ContentLength} bytes");
            Console.WriteLine($"File type: {metadata.Headers.ContentType}");
            Console.WriteLine($"Last modified: {metadata.LastModified}");
        }
        catch (AmazonS3Exception ex) when (ex.StatusCode ==
System.Net.HttpStatusCode.NotFound)
        {
            Console.WriteLine($"Error: File was not found in the bucket.");
            throw;
        }
    }

    private static void DisplayPresignedPostFields(CreatePresignedPostResponse
response)
    {
        Console.WriteLine($"Presigned POST URL: {response.Url}");
        Console.WriteLine("Form fields to include:");

        foreach (var field in response.Fields)
        {
            Console.WriteLine($"  {field.Key}: {field.Value}");
        }
    }

    /// <summary>
    /// Clean up resources created by the scenario.
    /// </summary>
    private static async Task CleanupAsync()
    {
        if (!string.IsNullOrEmpty(_bucketName))
        {
            Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");
            bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);

            if (result)
            {
                Console.WriteLine("Bucket deleted successfully");
            }
            else
            {
                Console.WriteLine("Failed to delete bucket - it may have been
already deleted");
```

```
            }
        }
    }
}
```

将您的项目迁移到 适用于 .NET 的 AWS SDK

本节提供有关版本 4 的迁移任务的信息 适用于 .NET 的 AWS SDK，以及有关如何执行这些任务的说明。有关迁移到早期版本的信息（例如，从 V2 迁移到 V3，或者从一个 V3 子版本迁移到另一个 V3 子版本），请参阅适用于 .NET 的 AWS SDK 开发者指南（版本 3）中的迁移项目。

另请参阅 新增内容 和 支持的平台。

主题

- 正在迁移到版本 4 的 适用于 .NET 的 AWS SDK
- 从 .NET Standard 1.3 迁移

正在迁移到版本 4 的 适用于 .NET 的 AWS SDK

适用于 .NET 的 AWS SDK 版本 4 (V4) 与 SDK 的版本 3 (V3) 相比有大量重大更改。本主题介绍版本 4 中的重大更改以及从 V3 迁移环境或代码可能需要完成的工作。有关 SDK 中其他值得注意的变更的更多信息，请参阅以下资源：

- 上的开发跟踪器问题:. GitHub https://github.com/aws/aws-sdk-net/issues/3362
- 博客文章 适用于 .NET 的 AWS SDK V4 的预览 1。
- 博客文章 适用于 .NET 的 AWS SDK V4 的预览 4。
- 博客文章 适用于 .NET 的 AWS SDK V4.0 的正式上市。

NET Framework。

.NET Framework 3.5 目标已从的 V4 中 适用于 .NET 的 AWS SDK删除。因此，软件开发工具包不再支持.NET Framework 3.5。此版本的 SDK 是针对.NET Framework 4.7.2 编译的，并在.NET 4.0 运行时中运行。有关更多信息，请参阅支持的平台。

值类型

在用于发出请求和响应的类中使用值类型的属性已更改为使用可为空的值类型。以下类型的属性已更改：

- `bool`已更改为 `bool?`
- `double`已更改为 `double?`

- `int`已更改为 `int?`
- `float`已更改为 `float?`
- `long`已更改为 `long?`
- `Datetime`已更改为 `Datetime?`

有关此更改的更多信息，请参阅博客文章 适用于 .NET 的 AWS SDK V4 的 Preview 1。

集合

现在，在用于发出请求和响应的类中使用集合的属性默认为`null`。因此，在尝试使用集合之前，您的代码需要验证集合是否为空。例如：

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

通过将设置为，可以恢复 V3 初始化集合的行为`Amazon.AWSConfigs.InitializeCollections`。`true`V3 中也存在此属性，供想要在升级到 V4 之前尝试此行为更改的用户使用。

有关此更改的更多信息，请参阅博客文章 适用于 .NET 的 AWS SDK V4 的 Preview 1。

AWS Security Token Service (STS)

- 区域终端节点

  使用依赖的凭证提供商时 AWS STS，呼叫始终使用区域终端节点。这与 SDK 的 V3 不同，后者在公共分区中运行时默认使用`us-east-1`区域，无论配置了哪个区域。

- **`StsRegionalEndpointsValue`枚举**

  该`StsRegionalEndpointsValue`枚举已从 Amazon.R untime 命名空间中删除。任何使用该枚举的代码都应该被删除。

- **`STSAssumeRoleAWSCredentials`** 类

已从 Amazon 上移除已弃用的 STS 代入角色凭证提供程序。STSAssumeRoleAWSCredentials SecurityToken命名空间。改为使用 AssumeRoleAWSCredentialsAmazon.Runtime。

与以下内容相关的更改 `ClientConfig`

Amazon.Runtime。 ClientConfig类是像 Amazons 3Config 这样的服务客户端配置类的基类。对该基类进行了以下更改。

- **默认重试模式**

  该`RetryMode`属性默认为`Standard`而不是`Legacy`。因此，该`Legacy`值已从 Amazon.Runtime 中删除。 RequestRetryMode枚举。
- **默认配置模式**

  该`DefaultConfigurationMode`属性默认为`Standard`而不是`Legacy`。因此，该`Legacy`值已从 Amazon.Runtime 中删除。 DefaultConfigurationMode枚举。
- 该**`ReadWriteTimeout`**财产

  已从除.NET Framework 4.7.2 之外的所有目标中删除了过时的`ReadWriteTimeout`属性。

AWSSDK.Extensions NETCore.Setup 软件包 NuGet

AWSSDK.Extensions NETCore.Setup NuGet 软件包已更新，以缓解 SDK V3 中存在的问题，并使该软件包对原生 AOT 来说是安全的。这些变化概述如下。有关详细信息，请参阅aws-sdk-net存储库中的 PR 3353。 GitHub

- **`DefaultClientConfig`** 类

  该`DefaultClientConfig`类不再继承自服务客户端配置基类 Amazon.Runtime。 ClientConfig。中的相关属性`ClientConfig`已`DefaultClientConfig`使用可为空的值类型进行复制。此更改使我们能够在将值复制到为服务客户端创建的配置`DefaultClientConfig`时检测何时设置了值。

  此更改的一个特别结果`DefaultClientConfig.HttpClientFactory`是，它在 V4 中不再可用。请改用 `AWSConfigs.HttpClientFactory`。有关更多信息，请参阅GitHub 问题 3790。

  此更改的另一个结果是，接受服务配置对象的`IConfiguration.GetAWSOptions`扩展方法的通用重载已被删除。应改用非通用重载，SDK 将自动处理填充特定于服务的设置。有关更多信息，请参阅GitHub 问题 3866。

- 原生 AOT

  该软件包中添加了一种用于创建使用 C# 11 静态接口方法的服务客户端的新机制。此更改无需加载程序集类型来创建服务客户端实例，包括对服务接口名称进行字符串操作以计算服务客户端类型，这与 Native AOT 不兼容。此更改仅适用于.NET 8 及更高版本；旧版本仍使用原始机制。

有关本指南中有关此软件包的更多信息，请参阅AWSSDK.Extensions.NETCore.Setup 和 iconfiguration。该软件包的源代码 GitHub 位于https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup。

`CookieSigner` 和 `UrlSigner`

Amazon 的`CookieSigner`和`UrlSigner`扩展 CloudFront 已移至名为 AWSSDK.Extensions 的单独扩展包中。 CloudFront.Signers。此更改是为了支持 OpenSSL 3 并依赖 BouncyCastle .Cryptography。

该软件包的源代码 GitHub 位于https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers。

DateTime 对阵世界标准时间 DateTime

某些 V3 类具有标记为 “已弃用” 或 “已过时” 的 DateTime 属性，以及备用 UTC DateTime 属性。在这些类中，过时的 DateTime 属性已被删除，UTC DateTime 属性的名称已更改为该 DateTime属性的原始名称。

以下是一些已实现此更改的类的示例。

- DescribeSpotPriceHistoryRequest:
  - 已删除过时的`StartTime`属性，并且该`StartTimeUtc`属性的名称已更改为 “StartTime”。
  - 已删除过时的`EndTime`属性，并且该`EndTimeUtc`属性的名称已更改为 “EndTime”。
- CreateFleetRequest
  - 已删除过时的`ValidFrom`属性，并且该`ValidFromUtc`属性的名称已更改为 “ValidFrom”。
  - 已删除过时的`ValidUntil`属性，并且该`ValidUntilUtc`属性的名称已更改为 “ValidUntil”。

如果应用程序使用的是原始的、过时的 DateTime属性，则此更改可能会导致偏移时间。使用 UTC DateTime 属性的代码将发生编译时错误。

DateTime 正在解析

该 DateTimeUnmarshaller 类已更新。该类一直在按本地时间解析和返回 DateTime字符串。在某些情况下，由于之前的更新，这些值会被转换回 UTC，但并非总是如此。现在，假定未编组的 DateTime 字符串为 UTC，并将按照 UTC 进行指定和解组。此更新包括以下行为更改。

基于该 DateTime 类的某些时间戳属性正在被解析为当地时间。其中包括用于时间戳的响应解组器和格式的列表时间戳以及。`TimestampFormat.ISO8601` `TimestampFormat.RFC822` DateTime 解析已更新，改为返回 UTC 时间。

`ConvertFromUnixEpochSeconds` 和 `ConvertFromUnixEpochMilliseconds`

ConvertFromUnixEpochSeconds和ConvertFromUnixEpochMilliseconds方法将 Unix 纪元秒转换为 DateTime结构，它们将 Unix 纪元时间作为本地时间而不是 UTC 时间返回。这些方法现在返回 UTC 时间。

日志记录

在 SDK 中启用登录功能的方式已针对 V4 进行了更新。登录控制台和系统诊断与 V3 的工作原理相同；也就是说，通过将AWSConfigs类的`LoggingConfig.LogTo`属性设置为`LoggingOptions.Console`或`LoggingOptions.SystemDiagnostics`。的LoggingOptions选项以及用于使用反射附加到内存中实例的 SDK 的内部逻辑`log4net`已被删除。`log4net`

为了将 SDK 登录到日志框架，需要使用单独的适配器包将 SDK 与日志框架连接起来。使用.extensions AWSSDK.Logging.Log4 包和.extens.Logg NetAdaptor ing。`log4net` AWSSDK ILogger适配器包适用于. `Microsoft.Extensions.Logging` 以下代码示例向您展示了如何在这两种情况下配置日志记录。

log4net 的配置示例

添加`AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet 软件包并从中调用静态`ConfigureAWSSDKLogging`方法`Log4NetAWSExtensions`。

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]
```

```
Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Microsoft.extens.Loggin

添加AWSSDK.Extensions.Logging.ILoggerAdaptor NuGet 软件包并从ILoggerFactory接口调用ConfigureAWSSDKLogging扩展方法。

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

对 HTTP 2 的支持

已添加对 HTTP 2 的 Support 以启用双向流式传输。有关更多信息，请参阅 对 HTTP 2 的支持。

单点登录

SSOAWSCredentials选项类SupportsGettingNewToken属性的默认值已从更改true为false。如果您的应用程序使用该SSOAWSCredentials类来获取 SSO 凭证，则可能需要将该Options.SupportsGettingNewToken属性设置为。true有关此配置的示例，请参阅中的代码示例仅使用 .NET 应用程序的 SSO 教程。有关更多信息，请参阅aws-sdk-net GitHub 存储库中的 PR 3737。

特定于 DynamoDB 的更改

以下更改特定于亚马逊 DynamoDB。其中许多都是突破性的变化。

有关在 V4 中对 DynamoDB 进行更改的更多信息，请参阅博客文章 V4 的 适用于 .NET 的 AWS SDK 预览 4。 适用于 .NET 的 AWS SDK

打开以查看项目

适用于 DynamoDB 的 SDK 中的 V4 更改解决了一些与可测试性有关的问题，但主要集中在高级库上：

- 代码DocumentModel中命名的 .NET 文档模型。
- 代码中命名的 DataModel.NET 对象持久化模型。

有关这些编程模式的详细信息，请参阅本指南DynamoDB中的。

文档模型：更新了模拟**IAmazonDynamoDB**界面的异常

在 SDK 版本 4 之前的文档模型中，如果使用模拟的 D IAmazonynamoDB 接口初始化表，它将返回。NullReferenceExceptionInvalidOperationException改为返回 SDK 的 V4。异步Table方法应该适用于模拟客户端，但是从中调用同步方法时，您可能仍会看到异常。.NET/Core/Standard

有关此变更的更多信息，请参阅上的 PR 3388。 GitHub

文档模型：**FromJson**和**ToJson**方法

D oc ument 类的FromJson和ToJson方法现在使用System.Text.Json而不是序列化，并且 LitJson 已从 SDK 的 V4 中删除。 LitJson 使用的一个好处System.Text.Json是，此解析器支持使用.NET Decimal 类型，该类型支持更高的数字浮点属性的精度。

对象持久化模型：该**DynamoDBOperationConfig**类

在对象持久化模型中，对共享的 Dynamo DBOperation Config 类进行了以下更改：

- 该类已被分成新的特定于操作的类 SaveConfig，例如LoadConfig、和。QueryConfig采用的方法DynamoDBOperationConfig已被标记为已过时，将来可能会被删除。

  有关此变更的更多信息，请参阅上的 PR 3421。 GitHub
- MetadataCachingMode和DisableFetchingTableMetadata属性已从类中移除。这些属性未包含在前面提到的新的特定于操作的类中。移除的属性是应在 DynamoDB 类的全局Context属性或 Dynam AWSConfigs o Config 类上指定的表级设置。DBContext

  有关此变更的更多信息，请参阅上的 PR 3422。 GitHub
- 该类不再继承自 Dynamo Confi DBContext g 类。这可以防止您将DynamoDBOperationConfig对象传递给 Dynamo 的构造函数DBContext，在该构造函数中，特定于操作的配置（例如OverrideTableName）中的某些属性不适用。

  有关此变更的更多信息，请参阅上的 PR 3422。 GitHub

对象持久化模型：多态性

D ynamo DBPolymorphic TypeAttribute 类已添加到对象持久化模型中。该类支持多态类型的序列化和反序列化。有关更多信息，请参阅上的 PR 3643。 GitHub

文档模型和对象持久化模型：可模拟的操作

添加了新的特定于操作的接口，允许客户模拟 DynamoDB 操作。IDynamoDBContext接口上的出厂方法已更新为返回新接口。

有关此变更的更多信息，请参阅上的 PR 3450。 GitHub

- 对象持久化模型
  - 通过IBatchGet和IMultiTableBatchGet接口模拟BatchGet操作。
  - 通过IBatchWrite和IMultiTableBatchWrite接口模拟BatchWrite操作。
  - 通过ITransactGet和IMultiTableTransactGet接口模拟TransactGet操作。
  - 通过ITransactWrite和IMultiTableTransactWrite接口模拟TransactWrite操作。
  - 通过IAsyncSearch界面进行模拟Scan和Query操作。
- 文档模型
  - 通过ITable接口模拟Table操作。
  - 通过ISearch界面进行模拟Scan和Query操作。
  - 通过IDocumentTransactWrite和IMultiTableDocumentTransactWrite接口模拟TransactWrite操作。
  - 通过IDocumentTransactGet和IMultiTableDocumentTransactGet接口模拟TransactGet操作。
  - 通过IDocumentBatchWrite和IMultiTableDocumentBatchWrite接口模拟BatchWrite操作。
  - 通过IDocumentBatchGet和IMultiTableDocumentBatchGet接口模拟BatchGet操作。

文档模型和对象持久化模型：对原生 AOT 的支持

原生 AOT 的一个限制是支持嵌套的.NET 类型。在某些情况下，.NET 编译器的修剪组件可能会忽略这些嵌套类型。在这种情况下，您可能会收到异常，例如：“System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.”

你可以通过在代码路径中的DynamicDependency某处添加一个来解决这个限制，告知修剪器对子类型的依赖性。保存的顶级.NET 类型的构造函数可能位于此处。以下代码示例向您展示了如何使用该DynamicDependency属性：

```
[DynamoDBTable("TestTable")]
```

```
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
    public string Name { get; set; }

    public SubType SubType { get; set; }
}

class SubType
{
    public string SubName { get; set; }
}
```

迪纳摩 DBStreams

Dynamo DBStreams 已从 AWSSDK.dynamoDB NuGet 软件包中移除，可在自己的软件包 AWSSDK.Dynamo 中使用DBStreams，并有自己的命名空间。`Amazon.DynamoDBStreams`

允许删除该**`TableNamePrefix`**值

现在，您可以在单个操作级别上删除 Dynamo DBContext Config 类中的`TableNamePrefix`属性值。有关此变更的更多信息，请参阅上的 PR 3476。 GitHub

`RetrieveDateTimeInUtc` 属性

对于 Dynamo DBContext Config 类，该`RetrieveDateTimeInUtc`属性的默认值已更改为。`true`

`DynamoDBContextTableNamePrefix` 属性

已从 AWSConfigsDynamoDB 类中移除该`DynamoDBContextTableNamePrefix`属性。用户应该打电话`AWSConfigsDynamoDB.Context.TableNamePrefix`而不是

特定于 EC2 的更改

以下更改特定于 Amazon EC2。它们中的大多数或全部都是重大变化。

打开以查看项目

`GetDecryptedPassword`

Amazon EC2 的`GetDecryptedPassword`扩展已移至一个名为.extension AWSSDKs.ec2 的单独扩展包中。 DecryptPassword。此更改是为了支持 OpenSSL 3 并依赖 BouncyCastle .Cryptography。

该软件包的源代码 GitHub 位于https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword。

对 Amazon EC2 的支持 IMDSv1

已移除对实例元数据服务版本 1 (IMDSv1) 的支持。从 IMDS 获取凭证和其他元数据时，SDK 的 V4 始终使用实例元数据服务版本 2 (IMDSv2)。有关 IMDS 的更多信息，请参阅 Amazon EC2 用户指南中的使用 IMDS。

已更改或移除的编程元素

- 整个`Amazon.EC2.Import`命名空间和代码已被删除。
- 整个`Amazon.EC2.Util`命名空间和代码已被删除，其中包括用于查找 Windows 版 EC2 的 AMIs AMI 实用程序。
- 已从IpPermission类中移除过时的`IpRanges`属性。改用`Ipv4Ranges`或`Ipv6Ranges`属性。
- 以下过时的字段已从EC2InstanceMetadata类中删除：`EC2_METADATA_SVCEC2_METADATA_ROOT`、`EC2_USERDATA_ROOT`、`EC2_DYNAMICDATA_ROOT`、和`EC2_APITOKEN_URL`。

特定于 S3 的更改

以下更改特定于 Amazon S3。它们中的大多数或全部都是重大变化。

打开以查看项目

AWS 区域 us-east-1

为`us-east-1`该区域配置的 Amazon S3 服务客户端无法再访问其他区域中的存储桶。必须使用为存储桶所在区域配置的 S3 服务客户端来访问存储桶。

有关此更改的更多信息，请参阅博客文章 V4 的 Pre 适用于 .NET 的 AWS SDK view 4。

S3 加密客户端

在Amazon.S3.Encryption命名空间中定义的 Amazon S3 加密客户端已从 AWSSDK.S3 软件包中删除。该客户端已移至自己的名为 Amazon.extensions.S3. Encryption 的软件包中，其文档位于。https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html有关迁移的更多信息，请参阅the section called “S3 加密客户端迁移（V1 到 V2）”。有关 S3 加密的更多信息，请参阅 Amazon S3 加密客户端开发人员指南中的支持的加密算法。

适用于 S3 的标记指令 **CopyObject**

该TaggingDirective属性已作为该CopyObjectRequest类的公共属性公开，AmazonS3Client.CopyObject方法使用该属性。此属性对应于CopyObject操作中定义的 Amazon S3 x-amz-tagging-directive 参数。

标记指令不再自动设置为 COPY。如果开发人员未指定标记指令，则 S3 后端会自动假设该指令是 COPY，但是如果开发人员明确将该属性设置为空，则根本不会设置该值。

S3 配置的**UseArnRegion**属性

Amazon.s3.amazons3Config 类的UseArnRegion属性已更新，因此AWS_S3_USE_ARN_REGION环境变量优先于共享文件中的设置。s3_use_arn_region AWS config有关这些变量和设置的更多信息，请参阅AWS SDKs 和工具参考指南中的设置参考。

CopyObject和**CopyPart**方法的前导斜杠

Amazon S3 CopyObject 和CopyPart方法的前导斜线将不再被修剪。该DisableTrimmingLeadingSlash属性已从CopyObjectRequest和CopyPartRequest类中移除。

这些**DoesS3BucketExist...**方法

已从实现 AmazonS3 接口的 Amazons3uTil 类中删除了过时的DoesS3BucketExist和DoesS3BucketExistAsync方法。ICore这些方法之所以被删除，是因为它们始终使用 HTTP。改用 doess3 BucketExist V2 和 doesS3 V2A syn c BucketExist。

SDK 总是使用 sigv4

的版本 4 适用于 .NET 的 AWS SDK 始终使用 AWS 签名版本 4 (Sigv4) 对请求进行签名。此更改会导致以下相关更改：

- AWSConfigsS3 类的UseSignatureVersion4属性已被删除。

- Amazon.Runtime 的SignatureVersion财产。 ClientConfig类已被删除。为了向后兼容，Amazon S3 仅使用此属性。
- 该RegionEndpoint.Endpoint类已被删除。这包括用于覆盖 Amazon S3 签名版本的SignatureVersionOverride属性。请改用特定于服务的client.DetermineServiceOperationEndPoint()方法。
- 更新了 Amazons3util 的方法。 PostUpload和 S3 PostUploadSignedPolicy。 GetSignedPolicy使用 sigv4。因此，该S3PostUploadSignedPolicy.GetSignedPolicyV4方法已被删除，因为GetSignedPolicy现在执行的功能相同。此外，GetSignedPolicy还为区域终端节点提供了第三个参数。

GetACL和PutACL方法

Amazons3Client 类的GetACL和PutACL方法已被标记为已过时。要访问这些方法的功能，请改用以下新方法：GetBucketACLPutBucketACL、GetObjectACL、和PutObjectACL。

已移除过时的编程元素

Amazon S3 实现中的许多编程元素已从 SDK 的 V4 中删除，包括枚举值、类型、方法、命名空间等。下面列出了这些内容（如果之前尚未介绍），以及您可以采取哪些可能的措施来适应它们的删除。

- 该DisableMD5Stream属性已从TransferUtilityUploadRequest类中删除。请改用 DisableDefaultChecksumValidation 属性。

  此外，该CalculateContentMD5Header属性已从TransferUtilityUploadRequest类中删除。不再需要此属性，因为 SDK 默认会计算校验和。
- ServerSideEncryptionMethod和ServerSideEncryptionKeyManagementServiceKeyId属性已从CopyPartRequest类中删除。改用类中同名的属性，Amazons3 Client InitiateMultipartUploadRequest类的某些InitiateMultipartUpload...方法中使用了该属性。
- 该Expires属性已从GetObjectResponse类中删除。请改用 ExpiresString 属性。该字符串可能不是有效的时间戳格式，因此您的代码在转换为 a DateTime 时应使用该TryParse方法。
- 已从 S3Re gion 枚举中删除了过时的 AWS 区域 标识符。
- 该Prefix属性已从LifecycleRule类中删除。请改用 Filter 属性。

  此外，NoncurrentVersionTransition和Transition属性已从LifecycleRule类中删除。改用NoncurrentVersionTransitions和Transitions集合。
- 该Event属性已从TopicConfiguration类中删除。改用该Events集合。
- CalculateContentMD5标题属性。不再需要设置此属性，因为 SDK 默认会计算校验和。

- 该Bucket属性已从SelectObjectContentRequest类中删除。请改用 BucketName 属性。
- 该NumberOfUploadThreads属性已从TransferUtilityConfig类中删除。改用ConcurrentServiceRequests属性。

已移除的编程元素

从 SDK 的 V4 中删除了许多编程元素，包括枚举值、类型、方法、命名空间等。下面列出了这些内容（如果之前尚未介绍），以及您可以采取哪些可能的措施来适应它们的删除。

打开以查看项目

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 命名空间

Amazon.Auth.AccessControlPolicy.ActionIdentifiers命名空间已被删除。这包括在IdentityandAccessManagementActionIdentifiers类中定义的 IAM 操作标识符。应将使用这些操作标识符的代码更改为使用操作名称的字符串值。

有关更多信息，请参阅 the section called “通过 JSON 创建托管式策略” IAM 用户指南中的 JSON 策略概述。

ClientConfig 类

Amazon.Runtime。 ClientConfig类是像 Amazons 3Config 这样的服务客户端配置类的基类。以下编程元素已从该类中删除。

- DetermineServiceURL和DetermineDnsSuffix方法已被删除。改用服务客户端DetermineServiceOperationEndpoint的方法；例如，Amazons3Client。 DetermineServiceOperationEndpoint。
- 该ReadEntireResponse属性已被移除。请改用以下方法之一：
  - 的LogResponses财产AWSConfigs。 LoggingConfig班级。
  - 客户端配置的LogResponse属性；例如 Amazons 3Config。

Amazon.Runtime 命名空间

Amazon.Runtime 命名空间已更新如下：

- 过时的ECSTaskCredentials类已从命名空间中删除。改用GenericContainerCredentials提供商，它还支持 Amazon EKS Pod 身份。

- 已从命名空间中移除过时的StoredProfileAWSCredentials和StoredProfileCredentials类。使用网络SDKCredentials文件或 Amazon.Runtime 的SharedCredentialsFile类。 CredentialManagement改为命名空间。
- 该SSOAWSCredentials类的过时HasCachedAccessTokenAvailable方法已从命名空间中删除。
- 过时的EnvironmentAWSCredentials类已从命名空间中删除。请改用 AppConfigAWSCredentials 类。
- 过时的StoredProfileFederatedCredentials类已从命名空间中删除。改用联合AWSCredentials类。
- 以下过时的类已从命名空间中删除：EnvironmentVariableAWSEndpointDiscoveryEnabledProfileAWSEndpointDiscoveryEn和FallbackEndpointDiscoveryEnabledFactory。
- 已从AmazonWebServiceRequest类中移除过时的UseSigV4属性。请改用 SignatureVersion 属性。
- Amazon.Runtime.Internal.Util命名空间中的ProfileIniFile类有一个名为的重载方法。TryGetSection不支持out参数的方法版本nestedProperties已从类中删除。
- Amazon.Runtime.Internal.Auth命名空间中的过时EventBridgeSigner类已被删除。
- 已从WebServiceRequestEventArgs类中移除过时的Parameters字典属性。请改用 ParameteCollection 属性。

BouncyCastle

的源副本 BouncyCastle 已从 SDK 的 V4 中删除。

StoredProfileSAMLCredentials 类

Amazon 中的过时StoredProfileSAMLCredentials类。 SecurityToken.SAML 命名空间已被删除。请改用 Amazon.Runtime 命名空间中的联合AWSCredentials类。

AWSSDKUtils 类

以下方法已从AWSSDKUtils类中删除：ResolveResourcePathProtectEncodedSlashUrlEncode、和ConvertToUnixEpochMilliSeconds。

`ProfileManager` 类

该过时的`ProfileManager`类已从 Amazon.Util 命名空间中删除。使用 Amazon.Runtime 中的网络 SDKCredentials文件或SharedCredentialsFile类。 CredentialManagement改为命名空间。

`AWSConfigs` 类

已从AWSConfigs类中删除了以下过时的属性：`LoggingResponseLogging`、和`LogMetrics`。请改用 `LoggingConfig` 属性。

`ConditionFactory` 类

具有以下签名的方法已从ConditionFactory类中删除:`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. 改用该NewConditionUtc方法。

Amazon CloudFront 实用工具

已删除过时的`Amazon.CloudFront.Util`命名空间和`AmazonCloudFrontUtil`类。

AWS IoT

在该ListPrincipalThingsResponse类中，已移除对`NextToken`覆盖的旧版自定义，转而使用分页。

AWS Lambda

由于名称混乱，该AmazonLambdaClient类的以下`Invoke...`方法已被删除。

- 带有以下签名的V3方法已被删除:`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. 这是 SDK V3 中的一种同步方法。改用`InvokeResponse Invoke(InvokeRequest)`（用于同步处理）或`Task InvokeAsync(InvokeRequest, CancellationToken)`（用于异步处理）。
- 带有以下签名的V3方法已被删除:`Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`. 这是 SDK 的 V3 中的一种异步方法。请改用 `Task InvokeAsync(InvokeRequest, CancellationToken)`。

亚马逊 SageMaker 运行时

该PayloadPart类的过时构造函数已被删除。

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。此公告发布后， AWS 于2020年12月31日终止了对.NET St 适用于 .NET 的 AWS SDK andard 1.3 的支持。

AWS 在 2020 年 10 月 1 日之前，继续提供 适用于 .NET 的 AWS SDK 针对.NET Standard 1.3 的服务更新和安全修复。此后，.NET Standard 1.3 目标进入维护模式，这意味着没有发布任何新更新；仅 AWS 应用了关键错误修复和安全补丁。

2020年12月31日，对.NET Standard 1.3的支持终止 适用于 .NET 的 AWS SDK 了。在此日期之后，未应用错误修复或安全补丁。使用该目标构建的工件仍可供下载 NuGet。

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 适用于 .NET 的 AWS SDK。

如果您有疑问或疑虑，请联系 AWS Support。

为此提供安全保障 AWS 产品或服务

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。安全是您 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。

云安全 — AWS 负责保护运行 AWS 云中提供的所有服务的基础设施，并为您提供可以安全使用的服务。我们的安全责任是我们的重中之重 AWS，作为AWS 合规计划的一部分，我们的安全有效性由第三方审计师定期进行测试和验证。

云端安全 — 您的责任由您使用的 AWS 服务以及其他因素决定，包括数据的敏感度、贵组织的要求以及适用的法律和法规。

本 AWS 产品或服务通过其支持的特定亚马逊网络服务 (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规性计划 AWS 合规性工作范围内的AWS 服务。

主题

- 这方面的数据保护 AWS 产品或服务
- 身份和访问管理
- 为此进行合规性验证 AWS 产品或服务
- 对此的应变能力 AWS 产品或服务
- 为此提供的基础设施安全 AWS 产品或服务
- 在中强制执行最低的 TLS 版本 适用于 .NET 的 AWS SDK
- S3 加密客户端迁移（V1 到 V2）
- S3 加密客户端迁移（V2 到 V4）

这方面的数据保护 AWS 产品或服务

分 AWS 担责任模型适用于本 AWS 产品或服务中的数据保护。如本模型所述 AWS ，负责保护运行所有网络的全球基础设施 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。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。有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

出于数据保护的目的，我们建议您保护 AWS 账户 证书并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。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）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅AWS CloudTrail 用户指南中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 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 AWS 通过命令行接口或 API 进行访问时需要经过 FIPS 140-3 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 140-3 版》https://aws.amazon.com/compliance/fips/。

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）。这包括当您使用控制台、API 或 AWS SDK AWS 服务 使用本 AWS 产品或服务或其他产品或服务时。 AWS CLI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 URL，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。

身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可以帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 AWS 资源。IAM AWS 服务 是您可以免费使用的。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- 操作方法 AWS 服务 与 IAM 合作
- 问题排查 AWS 身份和访问权限

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在其中所做的工作 AWS。

服务用户 -如果您 AWS 服务 用于完成工作，则管理员会为您提供所需的凭据和权限。当您使用更多 AWS 功能来完成工作时，可能需要额外的权限。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。如果您无法访问中的某个功能 AWS，请参阅问题排查 AWS 身份和访问权限或 AWS 服务 您正在使用的用户指南。

服务管理员 — 如果您负责公司的 AWS 资源，则可能拥有完全访问权限 AWS。您的工作是确定您的服务用户应访问哪些 AWS 功能和资源。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。要详细了解贵公司如何使用 IAM 和 AWS，请参阅 AWS 服务 您正在使用的用户指南。

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。要查看您可以在 IAM 中使用的 AWS 基于身份的策略示例，请参阅 AWS 服务 您正在使用的用户指南。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过代入 IAM 角色进行身份验证。

您可以使用来自 AWS IAM Identity Center（IAM 身份中心）、单点登录身份验证或证书等身份源的 Google/Facebook 证书以联合身份身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 以加密方式签署请求。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先要使用一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

最佳做法是，要求人类用户使用联合身份提供商来 AWS 服务 使用临时证书进行访问。

联合身份是来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的证书进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户 AWS 使用联合身份提供商以使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 当委托人提出请求时，会评估这些政策。大多数策略以 JSON 文档的 AWS 形式存储在。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、群组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。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。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 的 AWS 托管策略。

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，但它们不使用 JSON 策略文档格式。

亚马逊 S3 和亚马逊 VPC 是支持 ACL 的服务示例。 AWS WAF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 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，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）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在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。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。要了解如何在涉及多种策略类型时 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

操作方法 AWS 服务 与 IAM 合作

要详细了解如何 AWS 服务 使用大多数 IAM 功能，请参阅 IAM 用户指南中的与 IA M 配合使用的AWS 服务。

要了解如何在 IAM 中 AWS 服务 使用特定服务，请参阅相关服务用户指南的安全部分。

问题排查 AWS 身份和访问权限

使用以下信息来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

- 我无权在中执行操作 AWS
- 我无权执行 iam：PassRole
- 我想允许我之外的人 AWS 账户 访问我的 AWS 资源

我无权在中执行操作 AWS

如果您收到错误提示，指明您无权执行某个操作，则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 awes:*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:
 awes: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 awes:*GetWidget* 操作访问 *my-example-widget* 资源。

如果需要帮助，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给该服务，而不是创建新的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，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我之外的人 AWS 账户 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。您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，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 AWS 支持这些功能，请参阅操作方法 AWS 服务 与 IAM 合作。
- 要了解如何提供对您所拥有资源的访问权限，请参阅 IAM 用户指南中的向您拥有的另一 AWS 账户个 IAM 用户提供访问权限。 AWS 账户
- 要了解如何向第三方提供对资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方 AWS 账户 拥有的提供访问权限。
- 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访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

为此进行合规性验证 AWS 产品或服务

我们的新 AWS 注册体验不是为受监管的工作负载设计的。如果您正在使用我们的全新 AWS 注册体验，但想 AWS 用于受监管的工作负载，则可以注册 AWS （高级）或激活 AWS 环境的高级功能。

要了解是否属于特定合规计划的范围，请参阅 “按合规计划 AWS 服务 中的 “按范围划分的合规计划”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用时的合规责任 AWS 服务 由数据的敏感性、贵公司的合规目标以及适用的法律法规决定。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

本 AWS 产品或服务通过其支持的特定亚马逊网络服务 (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规性计划 AWS 合规性工作范围内的AWS 服务。

对此的应变能力 AWS 产品或服务

AWS 全球基础设施围绕 AWS 区域 可用区构建。

AWS 区域 提供多个物理上分隔和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。

利用可用区，您可以设计和操作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设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

本 AWS 产品或服务通过其支持的特定亚马逊网络服务 (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规性计划 AWS 合规性工作范围内的AWS 服务。

为此提供的基础设施安全 AWS 产品或服务

本 AWS 产品或服务使用托管服务，因此受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅安全支柱架 AWS 构完善框架中的基础设施保护。

您使用 AWS 已发布的 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。
- 具有完美向前保密 (PFS) 的密码套件，例如 DHE（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 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。

本 AWS 产品或服务通过其支持的特定亚马逊网络服务 (AWS) 服务遵循分担责任模式。有关 AWS 服务安全信息，请参阅AWS 服务安全文档页面和合规性计划 AWS 合规性工作范围内的AWS 服务。

在中强制执行最低的 TLS 版本 适用于 .NET 的 AWS SDK

为了提高与 AWS 服务通信时的安全性，应将配置 适用于 .NET 的 AWS SDK 为使用 TLS 1.2 或更高版本。

适用于 .NET 的 AWS SDK 使用底层 .NET 运行时来确定要使用哪种安全协议。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。

TLS 1.3 是启用后量子加密的先决条件，这可能需要额外的操作或配置。要了解更多信息，请参阅《AWS 软件开发工具包和工具参考指南》中的启用混合后量子 TLS。

.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。 适用于 .NET 的 AWS SDK 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，则需要确保在注册表中启用 TLS 1.2 支持，如 #tls -12 中所述。 https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings 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 版本 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。

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。

如果你使用的.NET 框架运行时不使用操作系统设置（.NET Framework 3.5 到 4.5.2），则 适用于 .NET 的 AWS SDK 会尝试在支持的协议中添加对 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**
>
> 2024 年 8 月 15 日， 适用于 .NET 的 AWS SDK 终止了对.NET Framework 3.5 的支持，并将.NET Framework 的最低版本更改为 4.7.2。有关更多信息，请参阅博客文章 Important changes coming for .NET Framework 3.5 and 4.5 targets of the 适用于 .NET 的 AWS SDK。

如果你的应用程序在 Windows 7 SP1 或 Windows Server 2008 R2 SP1 上的更新.NET 框架上运行，则需要确保注册表中启用 TLS 1.2 支持，如 #tls -12 所述。 https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings 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，网址为https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls。

AWS Tools for PowerShell

AWS Tools for PowerShell将 适用于 .NET 的 AWS SDK 用于所有 AWS 服务呼叫。环境的行为取决于 PowerShell 你正在运行的 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 适用于 .NET 的 AWS SDK 和 .NET Framework 3.5 的说明进行操作。

> **Warning**
>
> 2024 年 8 月 15 日， 适用于 .NET 的 AWS SDK 终止了对.NET Framework 3.5 的支持，并将.NET Framework 的最低版本更改为 4.7.2。有关更多信息，请参阅博客文章 Important changes coming for .NET Framework 3.5 and 4.5 targets of the 适用于 .NET 的 AWS SDK。

- 使用 .NET Runtime 4.0 时，请按照之前提供的有关 适用于 .NET 的 AWS SDK 和 .NET Framework 4+ 的说明进行操作。

Windows PowerShell 6.0

Windows PowerShell 6.0 及更高版本在 .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
 $true).FrameworkName
```

按照之前提供的有关.NET Core 适用于 .NET 的 AWS SDK 及相关版本的说明进行操作。

Xamarin

对于 Xamarin，请参阅上的说明。 https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security 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：必须将 HttpClient 实现设置为安卓并将 SSL/TLS 实现设置为原生 TLS 1.2+。

对于 iOS

- 需要 iOS 7 或更高版本。
- 项目属性，iOS 版本：必须将 HttpClient实现设置为NSUrlSession。

对于 macOS

- 需要 macOS 10.9 或更高版本。
- 项目选项、构建、 Mac 构建：必须将 HttpClient 实现设置为NSUrlSession。

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” 中进行此设置，如中所述https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html。.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。

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。

Blazor WebAssembly 在 ASP.NET Core 3.1 的预览版中，只有支持的浏览器才支持 WebAssembly，如中所述。 https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms 在支持之前，所有主流浏览器都支持 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。

S3 加密客户端迁移（V1 到 V2）

> **Note**
>
> 如果您使用的是 V2 并想迁移到 V4，请参阅S3 加密客户端迁移（V2 到 V4）。

本主题介绍如何将您的应用程序从亚马逊简单存储服务 (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机而不必同时重新加密所有对象，提供了 V1-transitional “” 客户端。该客户端可以解密 V1 和 V2-encrypted 对象，但只能以格式加密对象。 V1-compatible V2 客户端可以解密 V1 和 V2-encrypted 对象（启用 V1 对象时），但只能以格式加密对象。 V2-compatible

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-transitional 客户机以阅读新格式。首先，更新您的应用程序，使其依赖于V1-transitional客户端，而不是 V1 客户端。该 V1-transitional 客户端使您的现有代码能够解密由新 V2 客户端编写的对象和以 V1-compatible 格式编写的对象。

   > **Note**
   >
   > 提供的 V1-transitional 客户端仅用于迁移目的。迁移到客户端后，继续升级到 V2 V1-transitional 客户端。

2. 将 V1-transitional 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-transitional 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使这些客户端能够解密以 V1-compatible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，在所有客户端都已迁移到 V2 且所有对象均已按 V2-compatible 格式加密或重新加密后，将 V2 安全配置文件设置为`V2`而不是。`V2AndLegacy`这可以防止对 V1-compatible格式化对象进行解密。

将现有客户端更新为 V1-transitional 客户机以阅读新格式

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。

该 V1-transitional 客户端使您的应用程序能够解密 V1 和 V2-encrypted 对象。这个客户端是Amazon.Extensions.S3.Encryption NuGet 软件包的一部分。要使用 V1-transitional 客户端，请在每个应用程序上执行以下步骤。

1. 对Amazon.Extensions.S3.Encryption软件包建立新的依赖关系。如果您的项目直接依赖于AWSSDK.S3或AWSSDK.KeyManagementService包，则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该新包一起使用。

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。

该 V1-transitional 客户端完全 API-compatible 使用 V1 客户端，因此无需更改其他代码。

将 V1-transitional 客户端迁移到 V2 客户端以编写新格式

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-encrypted 对象（如果配置为解密），但只能以格式加密对象。 V2-compatible

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

Pre-migration

```
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);
```

Post-migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new
 EncryptionMaterialsV2("1234abcd-12ab-34cd-56ef-1234567890ab", KmsType.KmsContext,
 encryptionContext);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

示例：对称算法（AES-CBC 到 AES-GCM 密钥换行）

`StorageMode` 可以是 `ObjectMetadata` 或 `InstructionFile`。

Pre-migration

```
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);
```

Post-migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
 SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

> **Note**
>
> 使用解密时 AES-GCM，请先将整个对象读取到最后，然后再开始使用解密后的数据。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 密钥换行）

`StorageMode` 可以是 `ObjectMetadata` 或 `InstructionFile`。

Pre-migration

```
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);
```

Post-migration

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterialsV2(asymmetricAlgorithm,
 AsymmetricAlgorithmType.RsaOaepSha1);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

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);
```

S3 加密客户端迁移（V2 到 V4）

> **Note**
>
> 如果您使用的是 V1 并想迁移到 V4，则必须先迁移到 V2。请参阅S3 加密客户端迁移（V1 到 V2）。

本主题介绍如何将您的应用程序从亚马逊简单存储服务 (S3) 加密客户端的版本 2 (V2) 迁移到版本 4 (V4)，并确保应用程序在整个迁移过程中的可用性。V4 使用 AES-GCM 密钥承诺进行内容加密，并引入了承诺策略，以增强抵御密钥替换攻击的安全性。

V4 客户端在Amazon.Extensions.S3.Encryption NuGet 软件包中可用。

> **Important**
>
> 重大更改：配置了**RequireEncryptRequireDecrypt**策略的 V4 客户端无法解密使用 V1 或 V2 客户端加密的对象。只有最新的 V2 客户端才能使用密钥承诺解密 V4-encrypted 对象。在迁移到该RequireEncryptRequireDecrypt策略之前，必须使用启用密钥承诺的 V4 客户端重新加密所有现有数据。

理解 V4 概念

V4 使用 AES-GCM 密钥承诺进行内容加密，并引入了增强加密数据保护的关键安全概念：

承诺政策

承诺策略控制加密客户端在加密和解密操作期间如何处理密钥承诺。V4 支持三种承诺策略：

`ForbidEncryptAllowDecrypt`

加密：无承诺

解密：允许未提交的对象

安全性：不强制执行承诺，可能允许篡改

兼容性：所有 V2 和 V4 实现都可以读取使用此策略加密的对象

`RequireEncryptAllowDecrypt`

加密：附密钥承诺

解密：允许提交和不提交对象

安全：保护新对象免受密钥替换攻击，旧对象仍可读

兼容性：只有 V4 支持此政策

`RequireEncryptRequireDecrypt`（V4 的默认值）

加密：附密钥承诺

解密：仅提交对象

安全：全面执行承诺以最大限度地提高安全性

兼容性：只有 V4 支持此政策

带有关键承诺的 AES GCM

V4 使用 AES-GCM 密钥承诺进行内容加密，从而增强安全性：

- 篡改保护：它通过加密方式将密钥绑定到加密数据来防止密钥替换攻击。
- 版本兼容性：使用密钥承诺加密的对象只能由 V4 客户端和更新版本解密。

> **Warning**
>
> 在生产环境中启用密钥承诺加密之前，请确保所有需要解密对象的应用程序都已升级到 V4 或更高版本，因为 V2 客户端已被弃用。

更新现有客户端以读取 V4 格式

V4 加密客户端使用旧版本的客户端不支持的加密算法。迁移的第一步是更新您的 V2 客户端，以便它们可以读取新的 V4 格式。

更新 NuGet 软件包依赖关系

更新您的应用程序以使用包含 V4 支持的最新版本的Amazon.Extensions.S3.Encryption NuGet 软件包。对每个应用程序执行以下步骤：

1. 更新到最新的Amazon.Extensions.S3.Encryption软件包。如果您的项目直接依赖于AWSSDK.S3或AWSSDK.KeyManagementService包，则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该新包一起使用。
2. 确保您的`using`语句引用正确的命名空间：

   ```
   using Amazon.Extensions.S3.Encryption;
   using Amazon.Extensions.S3.Encryption.Primitives;
   ```

3. 重新构建并重新部署您的应用程序。

您现有的 V2 客户端将继续使用更新后的软件包，并将能够解密由 V4 客户端加密的对象（取决于所使用的承诺政策）。

构建和部署应用程序

更新 NuGet 软件包依赖关系后：

1. 构建您的应用程序以确保正确解决所有依赖关系。
2. 在开发环境中测试您的应用程序，以验证现有功能是否可以继续运行。
3. 将更新后的应用程序部署到您的生产环境。

此更新使您现有的 V2 客户端能够解密将由 V4 客户端加密的对象，从而确保迁移过程中的兼容性。

迁移到 V4 客户端

更新现有客户端以读取新的加密格式后，您可以继续安全地更新应用程序以使用 V4 加密和解密客户端。V4 客户端通过密钥承诺提供增强的安全性，同时保持与现有加密对象的兼容性。

4 步迁移流程

从 V2 到 V4 的迁移遵循结构化的 4 步流程，以确保兼容性和安全性。每个步骤都代表一种特定的配置，在继续下一步之前，应将其部署到所有应用程序中。

1. 第 0 步：V2 客户端（起点）-您现有的 V2 实现
2. 第 1 步：兼容 V2 的 V4 -迁移到 V4 客户端，同时保持加密行为 V2-compatible
3. 第 2 步：使用密钥承诺写入的 V4 -使用密钥承诺开始加密，同时允许解密传统对象
4. 第 3 步：全面强制执行的 V4 -加密和解密都需要密钥承诺

第 0 步：V2 客户端（起点）

这代表您现有的 V2 客户端配置。此步骤演示了迁移前的起始状态。

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);
```

```
#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

第 1 步：兼容 V2 的 V4

迁移到 V4 客户端，同时保持与 V2 相同的行为。此步骤使用ForbidEncryptAllowDecrypt策略进行无承诺的加密，并允许解密所有对象。

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
```

```
    ContentBody = content
});
```

行为：无需承诺即可加密，可以解密提交和未提交的对象。与 V2 行为相同。

第 2 步：包含关键承诺写作的 V4

使用密钥承诺开始加密，同时保持解密的向后兼容性。此步骤使用`RequireEncryptAllowDecrypt`策略。

> **Warning**
>
> 在部署步骤 2 之前，请确保所有读取器已更新到步骤 1 或更高版本以处理密钥承诺加密。

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

行为：承诺加密，可以解密提交和未提交的对象。保护新对象免受密钥替换攻击。

第 3 步：全面执行的 V4

加密和解密都需要密钥承诺。此步骤使用RequireEncryptRequireDecrypt策略来最大限度地提高安全性。

> **Warning**
>
> 在部署步骤 3 之前，请确保系统中的所有对象均已使用密钥承诺重新加密（步骤 2）。此步骤将无法解密未经承诺的加密对象。

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

行为：承诺加密，仅解密承诺加密的对象。抵御密钥替换攻击的最大安全性。

其他配置示例

本节提供了在迁移期间使用不同选项配置 V4 客户端的其他示例。

启用旧版支持

要使 V4 客户端能够读取由 V1 和 V2 客户端加密的对象，请为客户端配置允许传统解密的承诺策略：

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

在迁移期间，如果您需要解密由旧客户端加密的对象，同时确保以增强安全性对新对象进行加密，请使用此配置。

配置存储方法

V4 支持两种加密元数据的存储方法。选择最适合您的用例的方法：

对象元数据（默认）

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

指令文件

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

InstructionFile当您需要保留对象元数据以用于其他目的或处理具有元数据大小限制的对象时使用。

配置承诺政策

根据您的安全要求和迁移阶段选择适当的承诺政策：

迁移阶段（V2 兼容性）

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

过渡阶段（推荐）

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

全面安全阶段

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

从初始迁移ForbidEncryptAllowDecrypt期间开始，进入过渡阶段，最后RequireEncryptRequireDecrypt在所有客户端都已升级且所有对象都经过承诺重新加密后使用。RequireEncryptAllowDecrypt

的特殊注意事项 适用于 .NET 的 AWS SDK

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

- 获取用于的组件 适用于 .NET 的 AWS SDK
- 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

获取用于的组件 适用于 .NET 的 AWS SDK

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。使用安装 AWSSDK 软件包 NuGet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. 的引用的推荐方法 适用于 .NET 的 AWS SDK。）

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-net472.zip-支持.NET 框架 4.5 及更高版本的程序集。
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，如安装 AWSSDK 程序集时不使用 NuGet中所述。

访问应用程序中的凭证和配置文件

使用凭据的首选方法是允许为您查找和检索凭证，如中所述凭证和配置文件解析。 适用于 .NET 的 AWS SDK

但是，您也可以将应用程序配置为主动检索配置文件和证书，然后在创建 AWS 服务客户端时明确使用这些凭据。

要主动检索个人资料和证书，请使用 Amazon.Runtime 中的类。 CredentialManagement命名空间。

- 要在使用凭据文件格式（默认位置的共享 AWS 凭 AWS 据文件或自定义凭据文件）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 商店中查找配置SDKCredentials文件，请使用 Net File 类。
- 要根据类属性的配置同时在凭证文件和 SDK Store 中进行搜索，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，而不是使用该`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 F AWSCredentialsactor y 类。

以下部分提供了这些类的示例。

课堂示例 CredentialProfileStoreChain

您可以使用或TryGetProfile方法从CredentialProfileStoreChain班级中获取凭证TryGetAWSCredentials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`为空或为空，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 凭据文件。
- 如果 `ProfilesLocation` 属性包含值，请搜索该属性中指定的凭证文件。

从 SDK 商店或共享 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

此示例还使用了 Amazon.Runtime。 AWSCredentials班级。

```
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 商店或共享 AWS 凭证文件中获取个人资料

此示例向您展示如何使用该 CredentialProfileStoreChain 类获取个人资料。这些凭据可以来自 SDK Store，也可以来自默认位置的共享 AWS 凭据文件。

此示例还使用了该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

此示例向您展示如何使用 CredentialProfileStoreChain 类获取证书。凭证来自使用 AWS 凭证文件格式但位于备用位置的文件。

此示例还使用了 Amazon.Runtime。 AWSCredentials班级。

```
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
```

```
}
```

类 SharedCredentialsFile 和 AWSCredentials工厂的示例

使用课程创建 Amazons3Client SharedCredentialsFile

此示例向您展示了如何在共享 AWS 凭证文件中查找个人资料，根据该配置文件创建 AWS 凭证，然后使用这些凭证创建 Amazons3C lient 对象。该示例使用该SharedCredentialsFile类。

此示例还使用CredentialProfile类和 Amazon.Runtime。 AWSCredentials班级。

```
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**
>
> Net SDKCredentials File 类的使用方式完全相同，唯一的不同是你要实例化一个新的 Net SDKCredentials File 对象而不是一个 SharedCredentialsFile 对象。

Unity 支持的特殊注意事项

在 U nity 应用程序中使用 适用于 .NET 的 AWS SDK 和.NET 标准 2.0 时，您的应用程序必须直接引用适用于 .NET 的 AWS SDK 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 适用于 .NET 的 AWS SDK 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。

- 你需要在 Unity 项目 DLLs 中加入以下内容，以及 AWSSDKfo DLLs r .Core 和你正在使用的其他 AWS 服务。从版本的 3.5.109 开始 适用于 .NET 的 AWS SDK，.NET 标准 ZIP 文件包含这些附加文件。 DLLs
  - Microsoft.Bcl。 AsyncInterfaces.ll
  - 系统. 运行时。 CompilerServices.unsafe.dl
  - System.Threading.Tasks.Extensions.dll

- 如果您使用 IL2CPP 构建 Unity 项目，则必须向 A `link.xml` sset 文件夹中添加文件以防止代码剥离。该`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该`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 de https://aws.amazon.com/blogs/veloper/ for-net-standard -2-0-referencing-the-aws-sdk-uwp/ 上的文章。from-unity-xamarin-or

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。

的 API 参考 适用于 .NET 的 AWS SDK

适用于 .NET 的 AWS SDK 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，请参阅适用于 .NET 的 AWS SDK API 参考。

除了上面给出的一般参考之外，引导式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。

关于 API 参考版本

前面介绍的 API 参考适用于 3.0 及更高版本的 适用于 .NET 的 AWS SDK。

有关从旧版本的 SDK 迁移的信息，请参阅 迁移您的项目

要查找早期版本的 SDK API 参考的已弃用内容，请参阅以下内容：

- 适用于 .NET 的 AWS SDK API 参考 V1（已弃用）
- 适用于 .NET 的 AWS SDK API 参考 V2（已弃用）

要查看已弃用的 适用于 .NET 的 AWS SDK API 参考资料，您需要将其解压缩并配置 Web 浏览器。接下来显示的说明是如何执行此操作的示例。它们基于在 Windows 系统上使用谷歌 Chrome 网络浏览器。使它们适应您的特定网络浏览器和操作系统。

查看已弃用的 API 参考版本 1

1. 下载已弃用的 适用于 .NET 的 AWS SDK API 参考版本 1 的 ZIP 文件。默认情况下，ZIP 文件的名称为`sdkfornet-api-ref_v1_deprecated.zip`。在这些说明中将使用该名称。
2. 将 ZIP 文件放在您选择的文件夹中。对于这些说明，假定文件夹名称为`C:\work\temp\api-refs\V1`。
3. 右键单击 ZIP 文件并选择 “全部提取”。接受默认位置，即这些说明`C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1`的默认位置。
4. 在`C:\work\temp\api-refs\V1`文件夹中为谷歌浏览器创建快捷方式。注意不要移动原始应用程序。
5. 在新快捷方式的属性中，设置以下字段：

- 目标："*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"

  对于--user-data-dir参数，请使用适合您的环境的文件夹名称。该文件夹不一定存在。
- 开始于：C:\work\temp\api-refs\V1

6. 为快捷方式指定一个适当的名称。
7. 打开快捷方式以查看旧的 API 参考。

查看已弃用的 API 参考版本 2

1. 下载已弃用的 适用于 .NET 的 AWS SDK API 参考版本 2 的 ZIP 文件。默认情况下，ZIP 文件的名称为sdkfornet-api-ref_v2_deprecated.zip。在这些说明中将使用该名称。
2. 将 ZIP 文件放在您选择的文件夹中。对于这些说明，假定文件夹名称为C:\work\temp\api-refs\V2。
3. 右键单击 ZIP 文件并选择 “全部提取”。接受默认位置，即这些说明C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2的默认位置。
4. 在C:\work\temp\api-refs\V2文件夹中为谷歌浏览器创建快捷方式。注意不要移动原始应用程序。
5. 在新快捷方式的属性中，设置以下字段：

   - 目标："*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"

     对于--user-data-dir参数，请使用适合您的环境的文件夹名称。该文件夹不一定存在。
   - 开始于：C:\work\temp\api-refs\V2
6. 为快捷方式指定一个适当的名称。
7. 打开快捷方式以查看旧的 API 参考。

文档历史记录

下表描述了自上次发布《适用于 .NET 的 AWS SDK 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
的性能最佳实践 适用于 .NET 的 AWS SDK	添加了有关重复使用服务客户端、处置响应、异步编程、配置垃圾收集和管理 HTTP 连接的指南。	2026 年 7 月 30 日
新增功能	添加了有关适用于 .NET CLI 的 AWS 部署工具 2.0 版的注释。	2025 年 10 月 17 日
AWS OpsWorks	AWS OpsWorks 已经到了生命的尽头。它已从文档中删除。	2025 年 9 月 16 日
新增功能	End-of-support 已经发布了针对V3的版本 适用于 .NET 的 AWS SDK。	2025 年 8 月 21 日
新增功能	分布式缓存提供商的正式可用性	2025 年 7 月 2 日
消息处理框架	.NET 的 AWS 消息处理框架已正式发布！	2025 年 5 月 5 日
新增功能	的第 4 版现 适用于 .NET 的 AWS SDK 已正式推出！有关将应用程序迁移到 V4 的信息，请参阅迁移到版本 4 。	2025 年 4 月 28 日
迁移到第 4 版 适用于 .NET 的 AWS SDK	包括有关将使用构建的应用程序迁移 适用于 .NET 的 AWS SDK 到版本 4 的预览信息。	2025 年 3 月 3 日

在 .NET AWS Aspire 中集成适用于 .NET 的 AWS SDK	中包含 AWS 了有关如何与 .NET Aspire 集成的信息。适用于 .NET 的 AWS SDK	2025 年 2 月 15 日
可观察性	宣布推出可观测性的 GA 版本	2025 年 2 月 10 日
新增功能	添加了关于完整性保护新默认行为的信息。	2025 年 1 月 15 日
S3 加密客户端 V4 迁移	添加了有关从 V2 迁移到 V4 的信息。	2024 年 12 月 9 日
新增功能	添加了有关 适用于 .NET 的 AWS SDK 版本 4 的第四个预览版本的信息。	2024 年 11 月 15 日